XPLA Designer

Hot Tools for Cool Silicon

© Philips Semiconductors 1996

Permission is hereby granted to freely distribute this document in printed and electronic formats in its entirety without modification[·]

Philips CPLD Technical Support

Philips Semiconductors Programmable Products Group
M/S 08
9201 Pan American Freeway NE
Albuquerque, New Mexico 87113

Phone: 1-888-COOL PLD (Toll free in the USA)
or
1-505-858-2996

Fax: 1-505-822-7804

email: coolpld@scs.philips.com

World Wide Web http://www.coolpld.com/

XPLA™, FZP™ (Fast Zero Power), and TotalCMOS™ are trademarks of Philips Semiconductors. All other trademarks are the property of their respective owners.

[·] *IMPORTANT NOTICE TO PURCHASERS: The Entire Physical Universe, Including This Product, May One Day Collapse Back into an Infinitesimally Small Space. Should Another Universe Subsequently Re-emerge, the Existence of This Product in That Universe Cannot Be Guaranteed.*

XPLA™ Designer version 2.1 User's Manual

CHAPTER 1 *INTRODUCTION TO XPLA DESIGNER*....8

Supported Devices....8
The CPLD Design Process....8
Design Definition....9
Functional Simulation....10
Device Fitting....11
Post Layout Simulation....11
Programming....12
The XPLA Help File....12
Technical Support....12

CHAPTER 2 *INSTALLING XPLA DESIGNER*....13

System Requirements....13
Installing From CD-ROM....13
Installing from Floppy Disk....14
Uninstalling XPLA Designer....14

CHAPTER 3 *XPLA™ ARCHITECTURE OVERVIEW*....15

XPLA™ Architecture....15
Logic Block Architecture....16
Macrocell Configuration....17
PLA Product Term Sharing....18
Simplistic Timing Model....19
Closing Remarks....20

CHAPTER 4 *GETTING STARTED WITH XPLA DESIGNER*....21

Design Description....21
Device Selection....23
Compile the Design....24
Fit the Design....24
Simulate the Design....26
Generate the stimulus....27
Using the Programmer....31

CHAPTER 5 *PHDL LANGUAGE OVERVIEW*....32

General Language Syntax....32
Identifiers....32
Reserved Words....33
Comments....33
Number Types....34
Order of Operations....35
PHDL File Format At a Glance....35
Header Section....37
Declarations Section....37
Logic Description Section....37
End Statement....37
Creating and Editing PHDL Source Files....37
Starting a New Design....38
Editing an Existing Design....38

Header Section *38*
Module Statement *38*
Title *39*
Property *39*
Declarations Section *39*
Constants *39*
Variables *39*
Macro Functions *40*
Signals *42*
Input Signals *42*
Output Signals *43*
Bi-Directional Signals *44*
Nodes *45*
Logic Description *47*
Dot Extensions *47*
Clocking *49*
Generating and Assigning Clocks *49*
Choosing Clock Polarity *51*
Resets and Presets *51*
Equations *53*
State Diagrams *55*
Truth Table *60*

CHAPTER 6 *SIMULATING XPLA™ DESIGNS* 64

Simulation process *64*
Description of Simulator menus *67*
New *68*
Open *68*
Save *68*
Save As *68*
Run *69*
Print *69*
Page Setup *69*
Exit *69*
Undo *70*
Add Signal *70*
Del Signal *71*
Add Event *71*
Del Event *72*
Add Bus *73*
Change Bus *73*
Add Clk *74*
Set Value *75*
Clear Status List *75*
Zoom Out *76*
Zoom In *76*
Full Screen *76*
Set Signal Height *78*
Transition Check *81*
Auto Save SCL on Run *82*
Drop Anchor *82*
Raise Anchor *84*
Simulation Length *88*
Practice Design Simulation *89*

SCL Brief *94*

CHAPTER 7 *CONTROLLING THE FITTING PROCESS* 99

Fitter Options *99*
Pin Pre-assignments *100*
Max P-terms per equation *100*
Activate D/T register synthesis *101*
Auto Node Collapse Mode *101*
Fitter Option Examples *101*
Constraining Pins and Nodes Example *101*
D/T Register Synthesis Example *102*
Device Utilization Verses Design Speed Example *106*

CHAPTER 8 *PHDL LANGUAGE REFERENCE* 109

case 109
declarations 110
else 111
end 113
endcase 114
equations 115
goto 116
if-then-else 117
istype 121
macro 123
module 124
node 124
pin 125
property 126
state 127
state_diagram 128
then 129
title 131
when-then-else 132
with 135

APPENDIX A *SCL: SIMULATION CONTROL LANGUAGE* 138

Theory of Operation *138*
XPLA Designer Simulator *139*
Simulation Control Language *140*
Input Format 141
General Information about Stimuli *142*
Subroutines *142*
Data Fields *143*
Variables *143*
Jumps and Labels *143*
SCL Keyword Summary *144*
SCL Keyword Definitions *144*
BUSI Switch Bus Line to Input State 144
BUSO Switch Bus Line to Output 145
CALL Subroutine Call 145
DATE End of data lines 145
DATS start of data lines 146
DATV assign Data Lines to a Variable 146
DECV Decode a Variable 147
END End of a Subroutine Declaration 148
F Finish 149
GT Goto Statement 149

IF IF Statement 150
IFV Condition Check on a Variable 151
INCR Increment a Variable with a Specified Value 151
IT Initialize To 152
LIST Listing of Signals with State "*" "#" 152
RET Return From Subroutine 153
S Sequence 153
SDC Set Data Counter 154
SETV Set a Variable to a Value 155
ST Set To 155
STAB Stability Check 156
ST DATA Set to DATA 156
SU Simulate Until 157
SUB Start of a Subroutine Declaration 157
SUNS Simulate Until the Network is Stable 158
TRAC Transition Check 159

APPENDIX B *DEVICE PIN OUT CONFIGURATIONS* 161

PZ3032/PZ5032 44 PIN PLCC 161
PZ3032/PZ5032 44 PIN TQFP 162
PZ3064/PZ5064 44 PIN PLCC 163
PZ3064/PZ5064 44 PIN TQFP 164
PZ3064/PZ5064 68 PIN PLCC 165
PZ3064/PZ5064 84 PIN PLCC 166
PZ3064/PZ5064 100 PIN PQFP 167
PZ3128/PZ5128 84 PIN PLCC 168
PZ3128/PZ5128 100 PIN PQFP 169
PZ3128/PZ5128 128 PIN LQFP 170
PZ3128/PZ5128 160 PIN PQFP 171

APPENDIX C *DESIGN EXAMPLES* 172

8-BIT SHIFT REGISTER 173
N-BIT ADDER 174
8-BIT ADDER - HIGH LEVEL 175
8-BIT ADDER - LOW LEVEL 176
4-BIT ADDER WITH CARRY IN AND CARRY OUT 177
BCD TO 7 SEGMENT DECODER 178
BIDIRECTIONAL I/O'S 179
8-BIT EQUALITY COMPARATOR - HIGH LEVEL IMPLEMENTATION 180
16-BIT COUNTER - LOW LEVEL 181
DUAL 16-BIT UP/DOWN/LOADABLE/ENABLED/RESETABLE COUNTERS' 182
3-BIT COUNTER 184
16-BIT GRAY CODE COUNTER 185
4-BIT GRAY CODE COUNTER 191
TIMER/COUNTER 193
16-BIT LOADABLE BINARY COUNTER - 2 REPS 195
16-BIT SYNCHRONOUS PRESCALED COUNTER - 2 REPS 198
SERIAL CRC GENERATOR USING A 16-BIT LFSR G(X) = X16 + X12 + X5 + 1 201
3 TO 8 DECODER 202
8-BIT LOADABLE DATA REGISTER 203
DRAM CONTROLLER 204
HIGH LEVEL IMPLEMENTATION OF 16 TO 8 MULTIPLEXER 211
DUAL 4 TO 1 MUX 212
DATAPATH CIRCUIT - TWO REPS 213
SMALL STATE MACHINE - 8 INPUTS, 8 REGISTERED OUTPUTS 215

ARITHMETIC CIRCUIT: 4 X 4 MULTIPLIER, 8-BIT ADDER, 8-BIT REGISTER 219
1-BIT ACCUMULATOR 221
MEMORY MAPPED I/O ADDRESS DECODER 224
8-BIT FAST PARITY GENERATOR 226
DRAM REFRESH COUNTER 227
THUNDERBIRD TAILIGHT CONTROL CIRCUIT 231
DENVER INTERNATIONAL AIR TRAFFIC CONTROLLER 233

APPENDIX D *ERROR MESSAGES* 235

WARNINGS 236
ERRORS 237
FITTER ERROR MESSAGES 244
Warnings *244*
Fatal Errors *245*
Logical Errors *247*
COMMAND AND INTERNAL ERRORS 247

APPENDIX E *SOFTWARE LICENSE AGREEMENT* 248

APPENDIX Z *ABOUT THIS MANUAL* 249

Chapter 1 *Introduction to XPLA Designer*

Welcome to the Philips XPLA Designer Manual! This manual provides the information you need to use XPLA Designer to successfully design with Philips CoolRunner CPLDs. The overall CPLD design process consists of five steps: design definition, functional simulation, device fitting, post layout (timing) simulation, and programming. The XPLA Designer provides the first four of these five design steps and also supports the final step by producing a JEDEC file which can be used by most industry programmers to configure the device. The XPLA Designer also produces Verilog and VHDL timing models which can be used in board level simulations. This manual provides an overview of the XPLA Architecture and a quick tutorial on how to use XPLA Designer to assist the reader in becoming familiar with Philips CPLDs and the XPLA Designer, respectively.

The explanations and examples used in this manual assume that the user has at least some familiarity with Microsoft Windows and CPLDs. This manual is intended to be both a reference manual for experienced users and a comprehensive instruction manual for beginners. Each section of this manual describes a step of the design process used when targeting Philips CPLDs. Examples in each section illustrate the concepts being discussed. Depending on your level of experience, you may wish to read through the entire manual or to study only specific sections.

Supported Devices

XPLA Designer software version 2.1 supports the following devices:

PZ3032 - 32 macrocell, 3.3 Volt CPLD

PZ5032 - 32 macrocell, 5 Volt CPLD

PZ3064 - 64 macrocell, 3.3 Volt CPLD (Preliminary Timing Model)

PZ5064 - 64 macrocell, 5 Volt CPLD (Preliminary Timing Model)

PZ3128 - 128 macrocell, 3.3 Volt CPLD (Preliminary Timing Model)

PZ5128 - 128 macrocell, 5 Volt CPLD (Preliminary Timing Model)

The CPLD Design Process

The Philips XPLA Designer is a stand-alone tool which includes all aspects of the design process including design definition, functional simulation, device fitting, post layout (timing) simulation and also produces a JEDEC file which can be used to program the device. Figure 1 shows the high-level CPLD design process. The following sections walk

the user through this design process. Each section first gives a general description of the development stage and then gives the reader a brief introduction of how to use the XPLA Designer to complete each stage. A detailed description of each design stage is given in subsequent chapters.

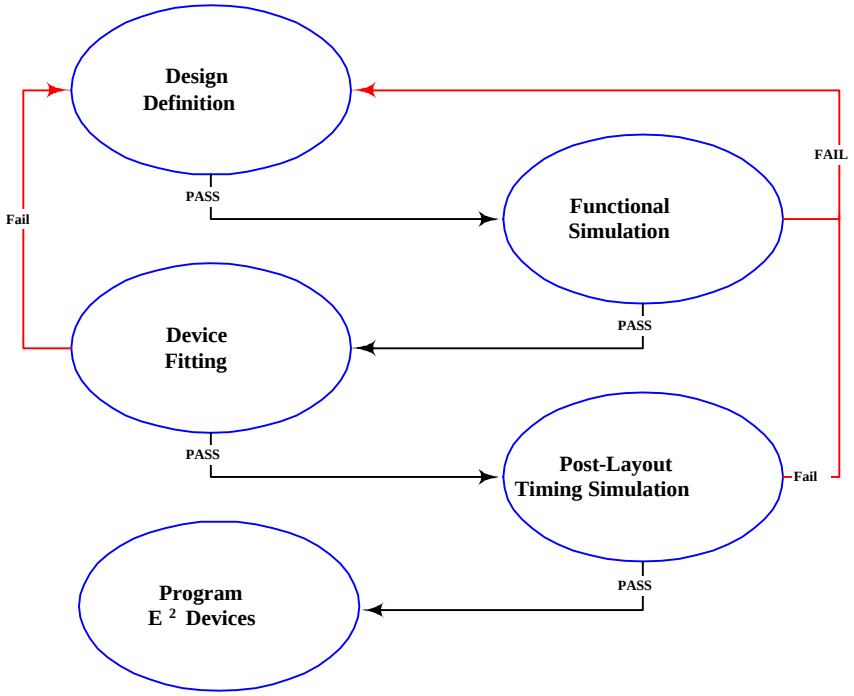


Figure 1

Design Definition

The design definition stage is where the design is actually created. Starting with the knowledge of what the design must do, the designer enters that information into the computer in various formats. Within the limitations of the design package, the designer may use various methods of design entry including: schematics, textual models, state diagrams, state machines, and boolean equations. To the computer, all of these formats are equivalent because the software will eventually link them all together. Having the ability to choose the design method gives designers a very powerful tool for their work. For example, designing a complex state machine using only registers and logic gates can be difficult; it may be easier to design it with an HDL (Hardware Description Language) that the computer can translate into those same registers and gates.

The XPLA Designer uses the PHDL (Philips Hardware Description Language) language to support the following design entry formats: boolean equations, state machines, and truth tables. A PHDL language overview and PHDL language reference are given in Chapters 5 and 8, respectively. A design can be created or selected by choosing the Design/New Design or the Design/Open Design command from the “Design pull-down Menu” on the XPLA Designer Interface shown in Figure 2. The design can be entered or

modified by selecting the “Edit Button” on the XPLA Designer Interface. Once the design has been entered, the PHDL must be compiled to check for syntax errors and to minimize the user’s logic. In order to activate the compiler, the “Compile Button” on the XPLA Designer Interface, as illustrated in Figure 2, must be selected.

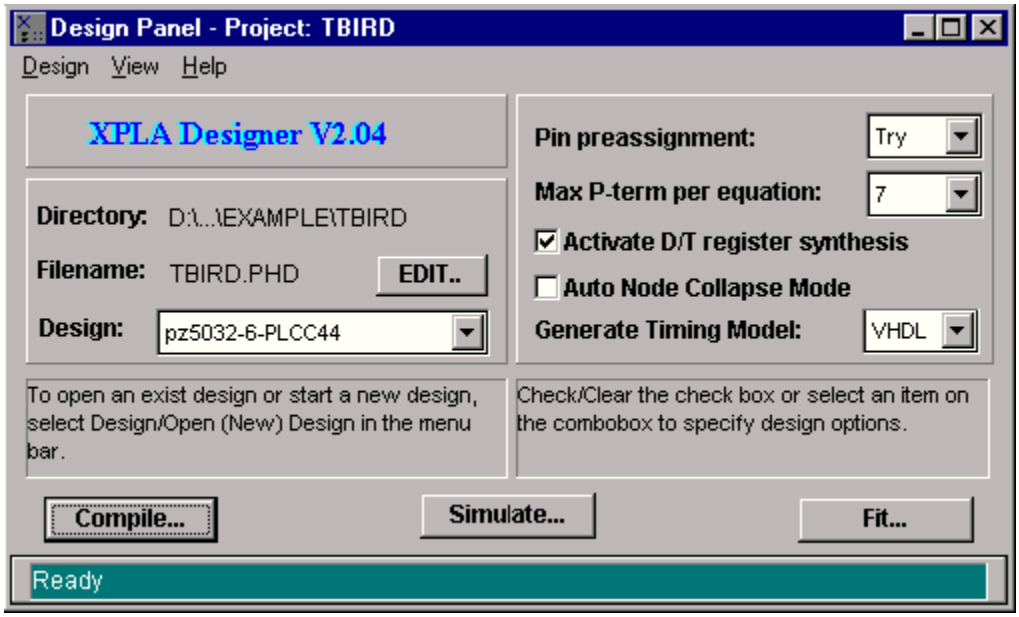

Design Panel - Project: TBIRD
Design View Help
XPLA Designer V2.04
Directory: D:\...\EXAMPLE\TBIRD
Filename: TBIRD.PHD
EDIT..
Design: pz5032-6-PLCC44
Pin preassignment: Try
Max P-term per equation: 7
Activate D/T register synthesis
Auto Node Collapse Mode
Generate Timing Model: VHDL
To open an exist design or start a new design, select Design/Open (New) Design in the menu bar.
Check/Clear the check box or select an item on the combobox to specify design options.
Compile...
Simulate...
Fit...
Ready

Figure 2

Functional Simulation

Functional simulation verifies that the *design* is performing as intended. This is different from verifying that the *part* is performing as intended. It checks only that the logical response of the design to particular input stimuli is correct, but does not check any physical parameters such as speed or power. For example, a simple design that adds two plus two will functionally simulate correctly if the output is four, even if it took hours for the output to appear.

When the XPLA Designer simulator is run after the design has been compiled but before the design has been fitted into the device, the simulator will act as a functional simulator. If the simulator is employed after the logic has been fit into the design, the XPLA Designer simulator will act as a timing simulator and use the actual timing parameters of the target device. The simulator can be activated by selecting the “Simulate Button” on the XPLA Designer Interface as shown in Figure 2. The simulation input stimuli is defined in a “Simulation Control Language” (SCL) file. A complete description of the XPLA Designer simulator can be found in Chapter 6.

Device Fitting

Device fitting is where software translates the design into a file format that a part programmer can understand and then attempts to fit the user's logic into the target device's resources. The file format varies depending on the part you are using. A device can be selected by highlighting the chosen device in the "Device pull-down Menu" on the XPLA Designer Interface. Once the sources have been compiled and the design is functioning properly, the XPLA Designer fitter can be employed by selecting the "Fit Button" from the XPLA Designer Interface.

The user can control the manner in which the fitter places the design into the device by using the following options on the XPLA Designer Interface:

- Pin Assignments
- Max P-term per Equation
- Activate D/T Register Synthesis
- Generate Timing Model

A complete description on how to control the device fitting process can be found in Chapter 7.

Post Layout Simulation

While the design may functionally simulate, the part may not function correctly due to physical limitations. For example, the part will not function correctly if you are using a 100 MHz clock, and there is a signal path in the part layout that takes more than 10 nanoseconds for the signal to reach the end of the path. To find this type of problem early in your design, you can do a second simulation that uses accurate delays from the datasheet specification of the devices to check the physical timing of the part. Like the functional simulation, this simulation also uses a test file that stimulates device inputs and records the outputs.

When fitting a design into the part, the XPLA Designer software generates a post-layout timing file. This file contains path delays for all signal routes based on real physical parameters for the selected CPLD. The XPLA Designer simulation uses this timing file to verify that the design will function correctly at the required frequency once it is programmed into the actual part. When the XPLA Designer simulator is employed after the design has been compiled and fitted into the device, the simulator will act as a timing simulator and use the actual timing parameters of the target device. The simulator can be activated by selecting the "Simulate Button" on the XPLA Designer Interface as shown in Figure 1.2. The simulation input stimuli is defined in a "Simulation Control Language" (SCL) file. A complete description of the XPLA Designer simulator can be found in Chapter 6.

Programming

This final stage is generally done when all other steps have been completed and all design specifications have been met during the post-layout simulation. Designs that successfully fit into the selected CPLD are also translated into a JEDEC file for use in programming the part. The JEDEC file can be loaded into a part programmer which then configures the design into a part.

The JEDEC format is understood by many commercially available parts programmers. Refer to the users manual of the specific programmer you are using for JEDEC compatibility and programming information.

The XPLA Help File

Included with the Philips XPLA Designer is a help menu. The help menu can be activated via the "Help Pull-Down Menu" in the XPLA Designer Interface and contains most of the information available in this manual

Technical Support

No documentation ever written can cover every conceivable concern a customer may have. Applications Engineers are available who are dedicated to making you successful in using Philips CoolRunner CPLDs. Should you have any questions regarding this product or our CoolRunner CPLDs, **please** contact us for assistance via any of the methods listed below.

Philips Semiconductors
Programmable Products Group Applications
9201 Pan American Freeway NE
Mail Stop 08
Albuquerque, New Mexico 87113

Phone: 1-888-COOL PLD (Toll free in the USA)
or
1-505-858-2996

Fax: 1-505-822-7804

email: coolpld@scs.philips.com

World Wide Web http://www.coolpld.com/

Chapter 2 *Installing XPLA Designer*

System Requirements

It is recommended that your system have the following as a minimum for using XPLA Designer:

486 PC running at 33 MHz or better
8 Megabytes of RAM
6 Megabytes of free disk space
Microsoft Windows™ 3.1 (or later) and Win32s (included on the XPLA Designer CD-ROM)

or

Microsoft Windows95™ (recommended - see appendix Z for comments)

Installing From CD-ROM

Insert the XPLA Designer CD into your CD drive.
From the Windows™ Program Manager, select File/Run.
Enter <DRIVE>:cdsetup.exe in the command box, where <DRIVE> is the letter representing your CD. (Win95 users may find that the autostart feature automatically loads the install program)
Select OK.

The CD-ROM Install Program will appear on screen, along with instructions. The install program will install any or all of the system components you specify. Following the instructions on screen, the CD-ROM will install the following components:

- XPLA Designer version 2.1 CPLD Design Tool
- Electronic documentation for the software and device datasheets in Adobe Acrobat™ .pdf format.
- Adobe Acrobat™ Reader for Windows version 2.1
- Microsoft Win32s 32bit runtime extension for Windows™ 3.1 (or later) users. Windows™ 3.1 must either have Win32s already installed or install the version provided to run XPLA Designer. **This is not needed for Windows95™ users.**

Once XPLA Designer is installed, you may run the program by following the instructions in Chapter 4. If you get an error when trying to run XPLA Designer on Windows 3.1 (or later) for the first time, it is likely that you do not have a version of Win32s on your machine that is not new enough to run XPLA Designer. In this case, you will need to upgrade to the Win32s version included with XPLA Designer. Once Win32s has been

upgraded, try to run the program again. If you still get an error, contact Philips CPLD Applications for support.

Installing from Floppy Disk

XPLA Designer is *only* shipped on CD-ROM. However, installation can be accomplished via floppy if the user can 'borrow' access to a machine that has a CD-ROM and a 1.44Mb 3.5" floppy drive. In the root directory of the CD-ROM there is a directory named FLOPPY. In this directory there are sub-directories named DISK1, DISK2, DISK3, . . . and so on (currently through DISK6). Each of these floppies contains 1.44Mb (or less) of information in the form of an Info-ZIP (Pkzip compatible) archive that has been split into multiple segments that will fit on the floppys. The last directory also contains the Info-ZIP public domain unzip utility and a batch file that will re-combine the split segments. Installation from a floppy-only machine can be accomplished by copying all of the files in each DISK*n* directory to separate 1.44Mb floppy disks on a 'borrowed' machine that has a CD-ROM, then copying the contents of all of the floppies to a single temporary directory (i.e. c:\tmp_xpla) onto the hard disk of the machine that has no CD-ROM drive. Then change directory to the temporary directory (i.e. cd c:\tmp_xpla). From a DOS command line, type **splice.bat.** This will copy all of the segments into a single zip file called XPLA.ZIP. Delete the separate split files, we're done with them - **del *.s*.** Next unzip the files and the subdirectories contained in the zip file - **unzip XPLA.ZIP**. Finally, from Windows run cdsetup.exe, which will invoke the install program - as detailed above. When the install is completed, all of the files and sub-directories in the temporary directory (i.e. c:\tmp_xpla) may be deleted. This approach requires 25Mb of free space on your hard disk..

Uninstalling XPLA Designer

In the event that you would like to completely remove XPLA Designer from your system, do the following:

Delete the <DRIVE>\XPLA directory and all its contents (DRIVE is the drive where you elected to install XPLA Designer).
Delete any design files from any directories where you have them stored.
Delete the file xplayer.ini from the Windows™ root directory.

Chapter 3 *XPLA™ Architecture Overview*

This chapter gives an overview of the XPLA™ architecture implemented in the Philips CoolRunner CPLD product family. The CoolRunner CPLD family combines a unique power saving design technique with a next generation architecture. The CoolRunner CPLD family includes devices ranging from 32 to 128 macrocells. This chapter focuses on the XPLA™ architecture and does not discuss the unique Fast Zero Power (FZP™) design technique used to implement these devices. The following sections give the CoolRunner features, an explanation of the XPLA™ architecture, and the advantages that these architectural features provide.

XPLA™ Architecture

Figure 3 gives a high level block diagram of the XPLA™ architecture. The XPLA™ architecture consist of Logic Blocks which are interconnected by a Zero-power Interconnect Array (ZIA). The ZIA is a virtual crosspoint switch. Each Logic Block is essentially a 36V16 device with 36 inputs from the ZIA and 16 macrocells. Each Logic Block also provides 32 ZIA feedback paths from the macrocells and I/O pins. The number of Logic Blocks contained within a device determines the macrocell count of the device. For example, devices containing 2, 4, and 8 Logic Blocks are 32, 64, and 128 macrocell devices, respectively.

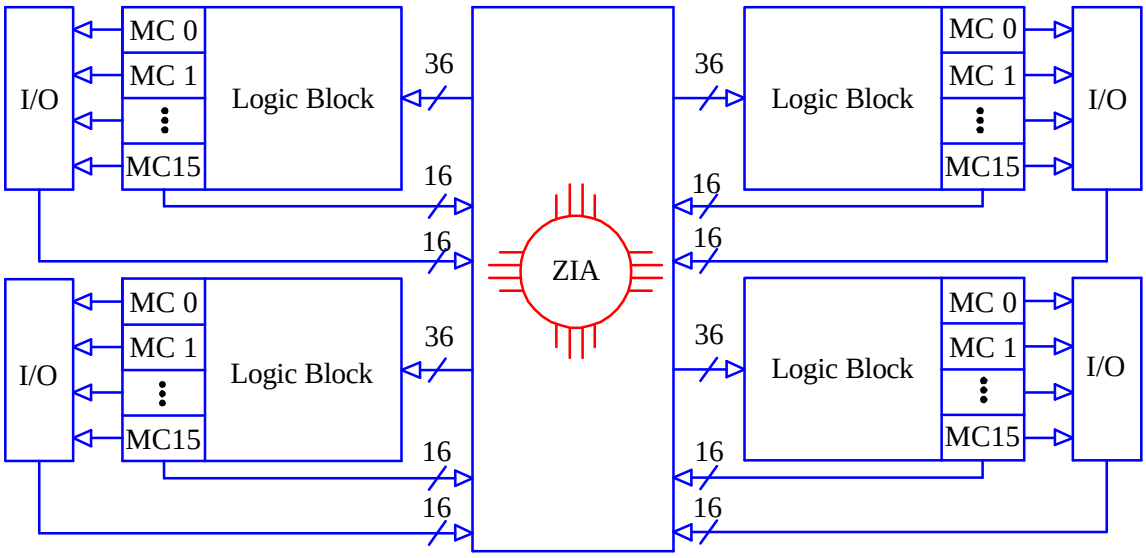


Figure 3

From this point of view, this architecture looks like many other CPLD architectures. What makes the CoolRunner family unique is what's inside each Logic Block and the design

technique used to implement these Logic Blocks. The contents of the Logic Block will be described next.

Logic Block Architecture

Figure 4 illustrates the Logic Block Architecture. Each Logic Block contains Control Terms, a PAL Array, a PLA Array, and 16 macrocells. The 6 Control Terms can individually be configured as either AND or SUM product terms and are used to control the preset/reset and output enables of the 16 macrocell's flip-flops. The PAL Array consists of a programmable AND array with a fixed OR array while the PLA array consist of a programmable AND array with a programmable OR array. The PAL array provides a high speed path through the array while the PLA array provides increased product term density.

Each macrocell has 5 dedicated product terms from the PAL array. If a macrocell needs more than 5 product terms, it simply gets the additional product terms from the PLA array. The PLA array consists of 32 product terms which are available for use by all 16 macrocells. For the 5V PZ5032 the additional propagation delay incurred by a macrocell using 1 or all 32 PLA product terms is just 2ns. So the total pin-to-pin Tpd for the PZ5032 using 6 to 37 product terms is 8ns (6ns for the PAL + 2ns for the PLA).

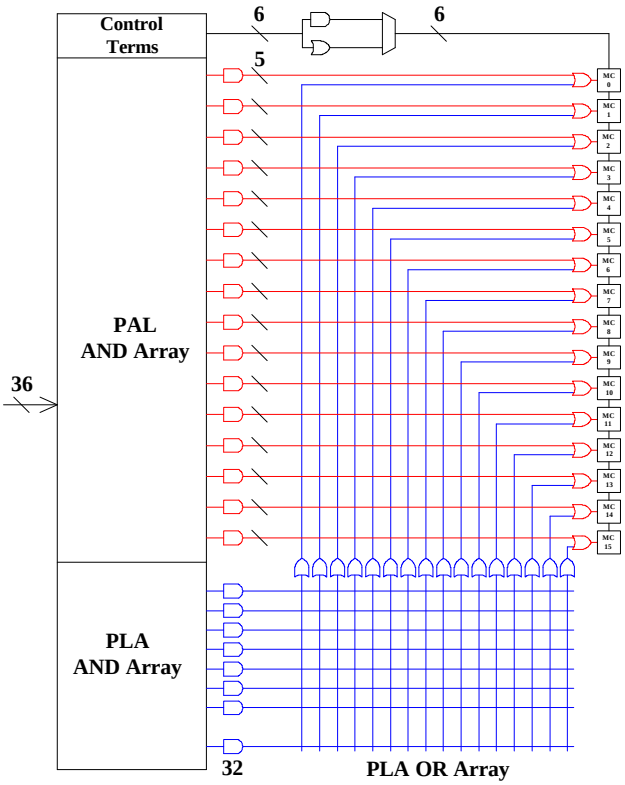


Figure 4

The XPLA™ architecture is very accommodating for implementing last minute design changes. In fact, 16 million worst case designs (designs which used all of the I/O Pins and all of the Macrocells) were implemented in the PZ5032 with fixed pins & macrocells and all but 30 designs were able to route. Therefore 99.998% of these worst case designs were able to route with the pins fixed after the design was changed.

The reason why the XPLA™ architecture accommodates last minute design changes is because the PAL product terms are dedicated to a given macrocell and in addition there is a free pool of 32 PLA product terms which can be used by any of the 16 macrocells. If a macrocell uses less than 5 product terms and the design change requires a total of 5 product terms, the design is guaranteed to fit because the 5 PAL product terms are dedicated to each macrocell. There is no borrowing between macrocells. Borrowing is a nice feature until the macrocell whose product terms were borrowed wants its product terms back because of a last minute design change. If a design change requires more than 5 product terms, unused PLA product terms are used by the macrocell. In an average design, less than 20 PLA product terms are used so there are typically 12 PLA product terms available to implement last minute design changes.

Macrocell Configuration

Figure 5 shows the architecture of the macrocell used in the CoolRunner family. The macrocell consists of a flip-flop which can be configured as either a D or T type. A D-Type flip-flop is generally more useful for implementing state machines and data buffering. A T-Type flip-flop is generally more useful in implementing counters. All CoolRunner family members provide both synchronous and asynchronous clocking and provide the ability to clock off either the falling or rising edges of these clocks. These devices are designed such that the skew between the rising and falling edges of a clock are minimized for clocking integrity. There are 2 clocks (CLK0 and CLK1) available on the PZ3032/PZ5032 devices and 4 clocks (CLK0 through CLK3) available in the PZ3064/PZ5064 and PZ3128/PZ5128 devices. Clock 0 (CLK0) in each of these devices is designated as the “synchronous” clock and must be driven by an external source. Clocks 1, 2, and 3 (CLK1, CLK2, and CLK3) can either be used as a synchronous clock (driven by an external source) or as a asynchronous clock (driven by a macrocell equation).

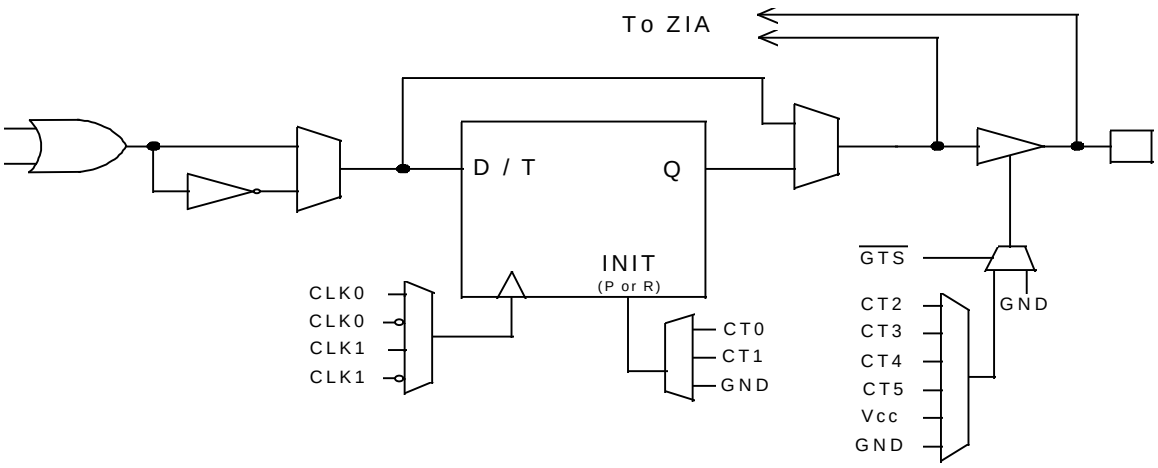


Figure 5

Two of the control terms (CT0 and CT1) are used to control the Preset/Reset of the macrocell's flip-flop. The Preset/Reset feature for each macrocell can also be disabled. The other 4 control terms (CT2-CT5) can be used to control the Output Enable of the macrocell's Output Buffers. The reason why there are so many control terms dedicated for the output enable of the macrocell is to insure that all CoolRunner devices are PCI compliant. The macrocell's output buffers can also be always enabled or disabled. All CoolRunner devices also provide a Global Three-State (GTS*) pin which, when pulled low, will three-state all the outputs of the device. This pin is provided to support "In-Circuit Testing" or "Bed-of Nails Testing".

There are two feedback paths to the ZIA; one from the macrocell and one from the I/O pin. The ZIA feedback path before the output buffer is the macrocell feedback path while the ZIA feedback path after the output buffer is the I/O pin ZIA path. When the macrocell is used as an output, the output buffer is enable and the macrocell feedback path can be used to feedback the logic implemented in the macrocell. When the I/O pin is used as an input, the output buffer will be three-stated and the input signal will be fed into the ZIA via the I/O feedback path and the logic implemented in the buried macrocell can be fed back to the ZIA via the macrocell feedback path.

PLA Product Term Sharing

Another feature offered by the XPLA™ architecture which cannot be offered by other competing architectures is product term sharing. In address decode circuits, some state machines, and other types of designs there are product terms which are common to a number of macrocells. The XPLA™ architecture allows sharing of PLA product terms between macrocells as shown in Figure 6. In this example, it shows one PLA product term being shared by two macrocells. In this case, there is "effectively" 33 PLA product terms because one of them is shared between two macrocells. PLA product term sharing increases the "effective" density of the device and allows larger designs to fit in the same macrocell count device. If needed, all 16 macrocells could share all 32 PLA product terms.

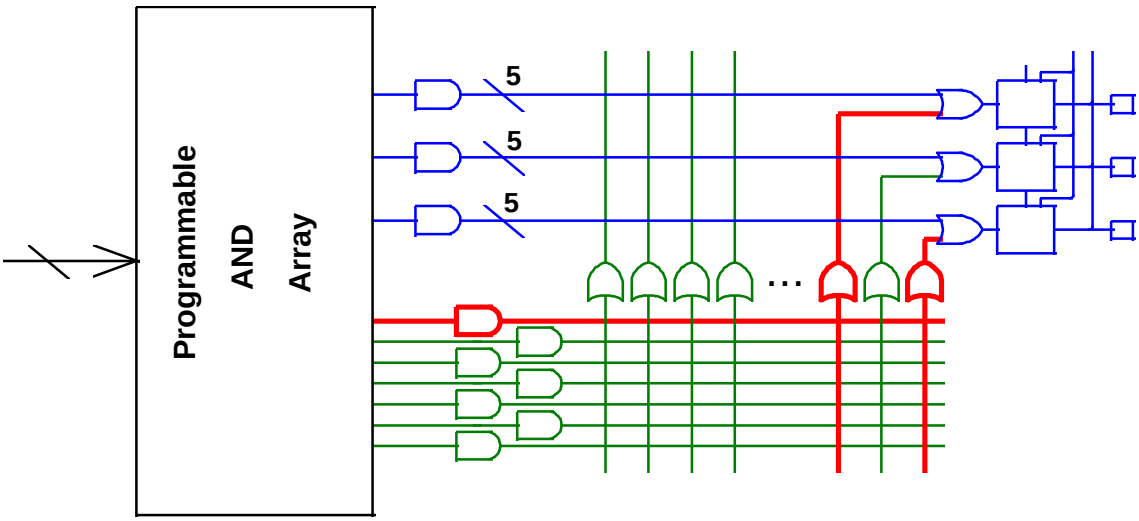


Figure 6

Simplistic Timing Model

Figure 7 shows the CoolRunner Timing Model. As one can see from this illustration, the CoolRunner timing model looks very much like a 22V10 timing model in that there are three main timing parameters including Tpd, Tsu, and Tco. In other competing architectures, the user may be able to fit the design into the CPLD but is not sure whether system timing requirements can be met until after the design has been fit into the device. This is because the timing models of competing architectures are very complex and include such things as: timing dependencies on the number of parallel expanders borrowed, sharable expanders, varying number of X and Y routing channels used, etc. In the XPLA™ architecture, the user knows up front whether the design will meet system timing requirements. This is due to the simplicity of the timing model. For example, in the PZ5032 device, the user knows up front that if a given output uses 5 product terms or less, the Tpd = 6ns, the Tsu = 4ns, and the Tco = 5.5ns. If an output is using 6 to 37 product terms, an additional 2ns is added to the Tpd and Tsu timing parameters to account for the time to propagate through the PLA array - this is the only variation in timing that exists when using the XPLA™ architecture!

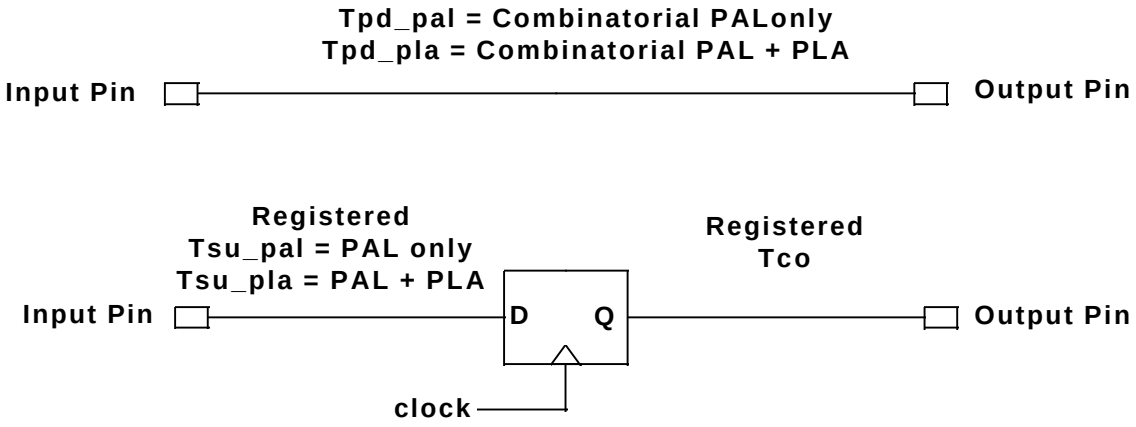


Figure 7

Closing Remarks

The XPLA™ architecture used in the Philips CoolRunner CPLD family combines a unique power saving design technique with a next generation architecture. For any questions concerning the architecture, please contact Philips Programmable Products technical support via the contact information detailed in Chapter 1.

Chapter 4 *Getting Started with XPLA Designer*

This chapter provides a tutorial which demonstrates how to create, compile, and simulate a design using XPLA Designer. The design uses the Philips Hardware Description Language to define a state machine.

Design Description

This getting started chapter targets a design called T-Bird Tail Lights into a Philips PZ3032 CPLD. This design is a state machine that simulates the blinkers and tail lights on Ford Thunderbird automobiles of the late 1960s (see Figure 8). The inputs to the state machine are clock (CLK), turn left (TL), turn right (TR), and brake. The outputs of the design are the left blinkers (L1, L2, and L3) and the right blinkers (R1, R2, and R3). When turning left or right, the blinkers sequence through three bulbs to indicate the direction of turn (see Figure 8). Starting with the inside bulb, three bulbs are sequentially lighted in the direction of the turn until all three are on; then all three go dark and the sequence starts again. When only the brake is applied, all six bulbs will light. If the brake and a turn signal are applied simultaneously, the turn signals function as above, but all three lights on the side opposite the direction of turn will light.

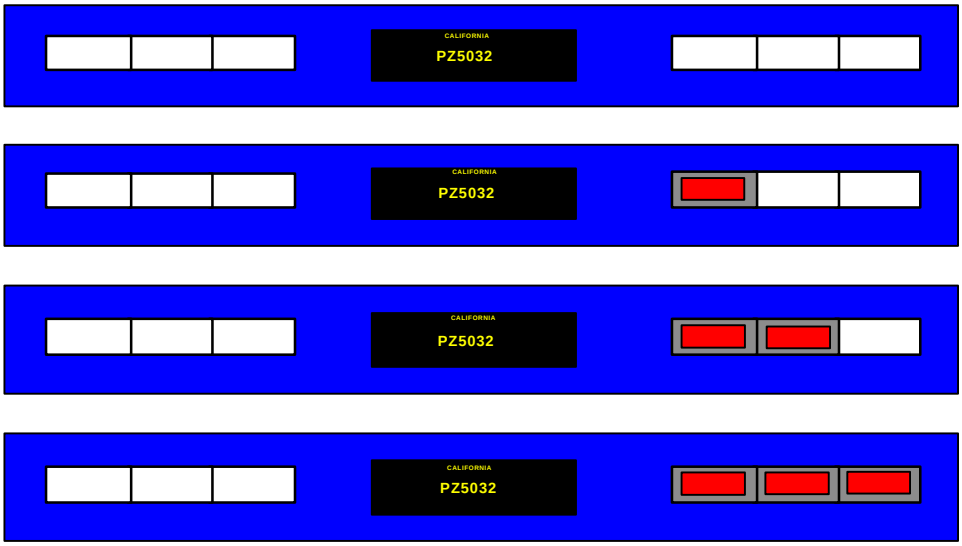


Figure 8

To begin, from the Windows Program Manager double-click on the XPLA Designer icon to invoke the tool. The XPLA Designer (Figure 9) user-interface consists of a menu bar at the top, design entry options on the left hand side, compilation options on the right hand side, and program execution options (compile, simulate, fit) at the bottom. Context sensitive help for the design entry and compilation option sections is provided just below the respective sections. Move the cursor over the fixed menus and read the context

sensitive help. The menu bar consists of three entries: **Design**, **View**, and **Help**. Left clicking on a menu bar entry causes a pull down menu to be displayed. The **Design** pull down menu allows the user to create a new design or edit an existing design. The **View** pull down menu allows a variety of files to be viewed. If a **View** file has not been generated, it appears in the list as "dimmed".

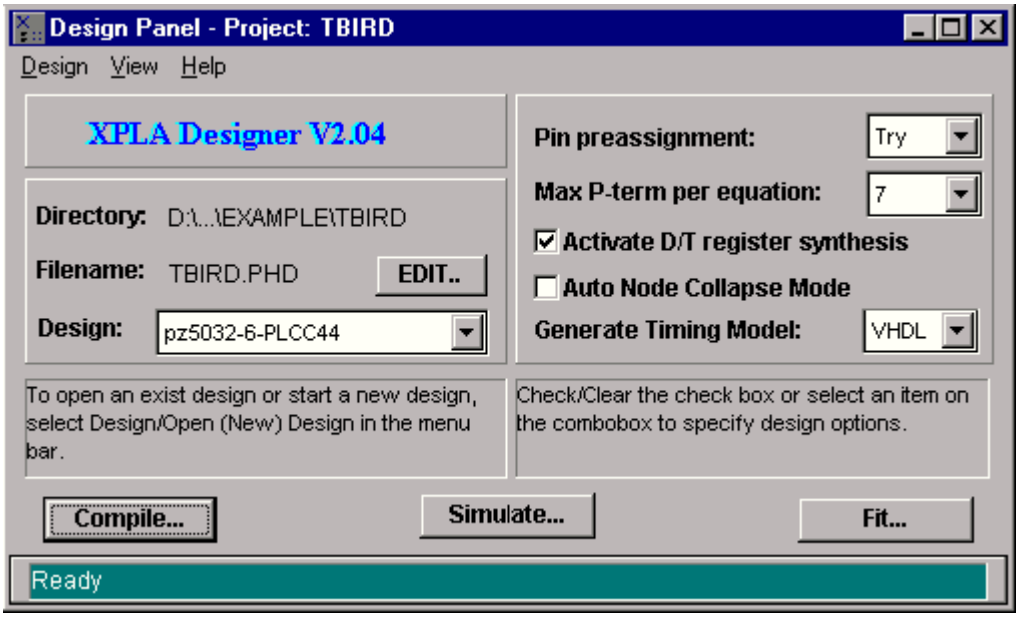

Design Panel - Project: TBIRD
Design View Help
XPLA Designer V2.04
Directory: D:\...\EXAMPLE\TBIRD
Filename: TBIRD.PHD
EDIT..
Design: pz5032-6-PLCC44
Pin preassignment: Try
Max P-term per equation: 7
Activate D/T register synthesis
Auto Node Collapse Mode
Generate Timing Model: VHDL
To open an exist design or start a new design, select Design/Open (New) Design in the menu bar.
Check/Clear the check box or select an item on the combobox to specify design options.
Compile...
Simulate...
Fit...
Ready

Figure 9

The tbird design is created in the \xpla\examples directory with the software installation. This tbird.phd file will be used in this tutorial. When either a new design file is created or an existing file is opened, a dialog box is displayed which allows the user to move to the design directory. See Figure 10.

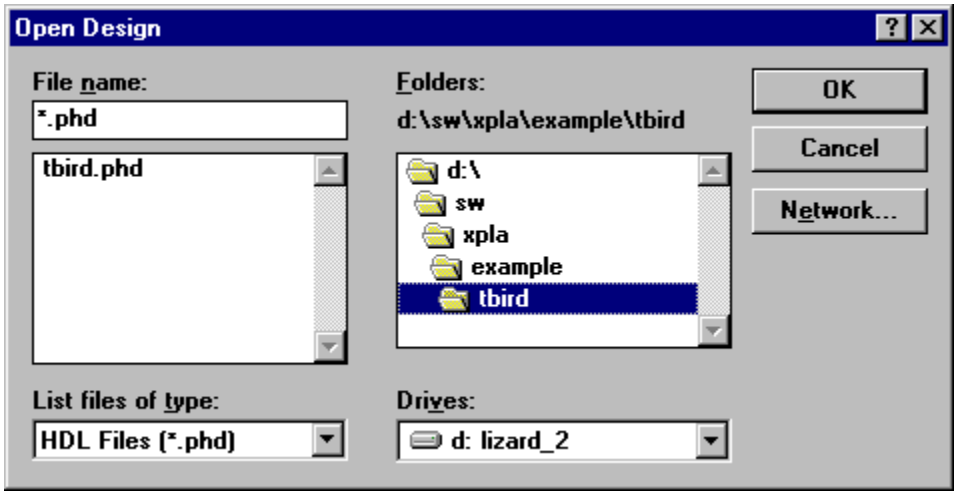


Figure 10

New design filenames must use a .phd filename extension. Existing design file extensions can use either phd, tt2 or blf extensions (tt2 & blf extensions are used only when importing a design from another front-end synthesis tool that supports PLA or BLIF format intermediate files). If creating a new design, use “tbird.phd” for the design filename. If using the source code provided, the PHDL code may be viewed by opening tbird.phd and then clicking the **Edit** button on the main screen. A text editor displays the PHDL code (Figure 11).

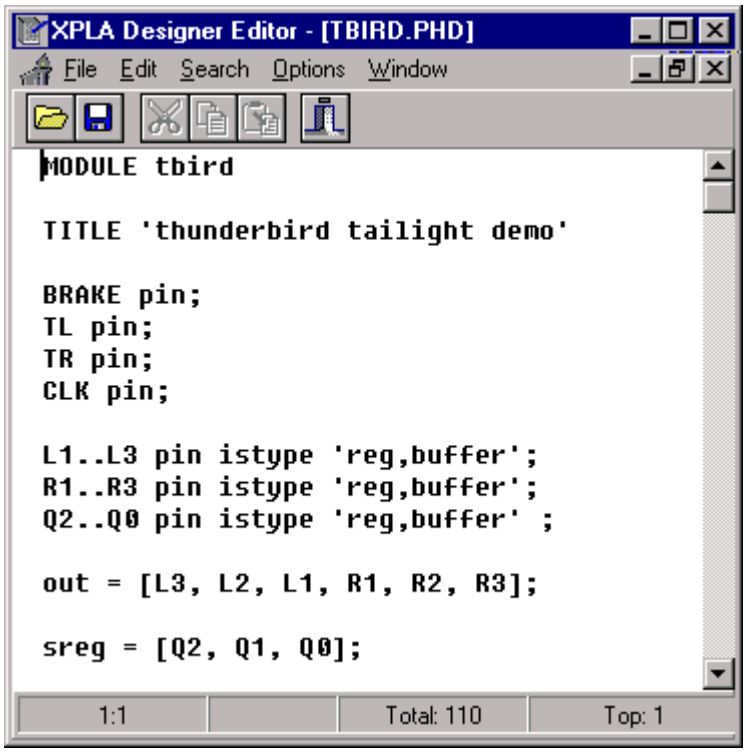


Figure 11

The text editor allows the user to enter new designs and/or edit existing designs. It provides several functions which are useful in the design flow, since it isn’t necessary to exit XPLA Designer when making design revisions. Design revisions can be overwritten or renamed with **File Save** or **File Save As** functions. The line number and column position are displayed in the lower left box. This is useful for finding errors in which the error message provides the line number of the code responsible for the error.

Device Selection

The Philips CoolRunner series consists of 32, 64, and 128 macrocell devices, in 3-volt and 5-volt versions, in a variety of speed grades and packages. View the device list by left clicking on the arrow adjacent to Devices. Scroll through the device list to see all devices displayed (Figure 12). The PZ3000 series devices operate from a 3-volt supply while the

PZ5000 series devices use a 5-volt supply. The number of macrocells is the last 3 digits in the 3000/5000 number. The speed grade is generally related to the pin to pin delay, and is designated next, followed by the package definition. This design targets an 8 ns PZ3032 in a 44-pin PLCC. Select the PZ3032-8-PLCC44.

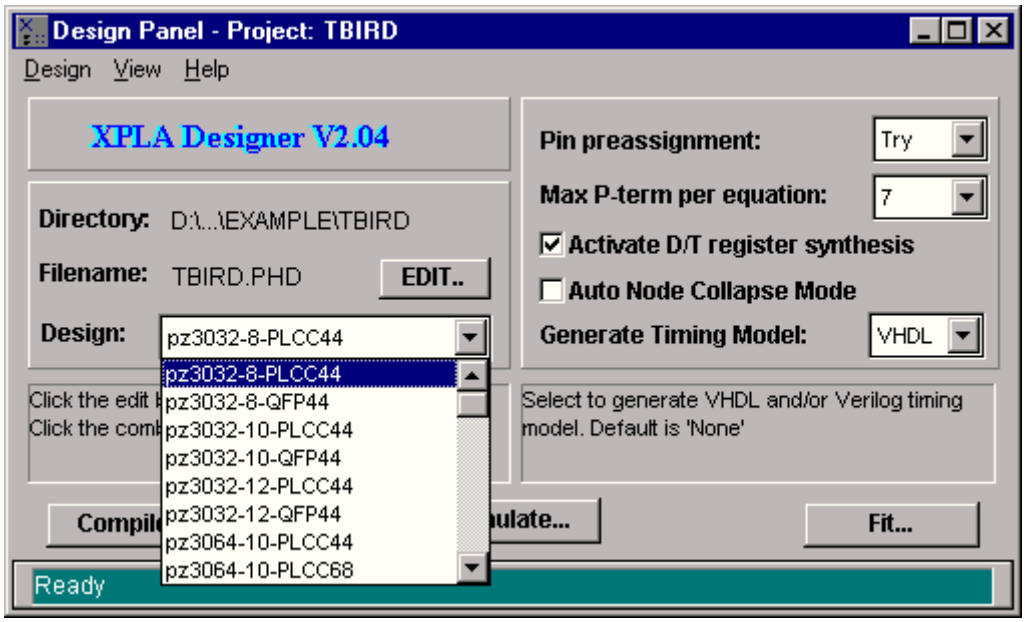


Figure 12

Compile the Design

Prior to compilation and fitting, select the compilation options. As seen in Figure 12, compilation options can be selected from the main interface screen via pull-down menu options and check boxes. The PZ3000/PZ5000 series can support from 5 to 37 P-terms/macrocell. There is an incremental pin to pin delay for macrocells in which the number of P-terms exceeds 5. Using the pull-down menu next to **MAX P-Terms per equation**, change the maximum P-term per equation to **5**. The user can define pin assignments in the phdl file. As a compilation option, the user can define the level of effort that the fitter should use to maintain the pinout assignments defined in the PHDL file. The options are Try, Keep, and Ignore. The pinout retention capability of the fitter is very high, so select **Keep** in the menu next to **Pin preassignment**. Click on the **Compile** button after setting the options to compile the design.

Fit the Design

After compilation, you can fit the design into the target device by clicking the **Fit** button. During the fitting process, XPLA Designer generates three files that contain information about the design. The "tbird.fit" file contains the fitter report (Figure 13) which indicates

the CPLD utilization and pinout. The “tbird.tim” file contains timing information. The “tbird.jed” file can be used with a device programmer such as Data I/O’s Unisite/2900/3900 or BP Microsystems to configure the CPLD.

The fitter report is shown in Figure 13. Examine the report by pulling down the **View** menu and selecting ‘**Fitter report’**, to see how many macrocells the design used, the percentage utilization and the pinout. Also, take a look at the timing report, a section of which is shown in Figure 14, by pulling down the **View** menu and selecting **‘Timing report’**. The timing specifications provided in this report accurately reflect the AC parameters in the data sheet.

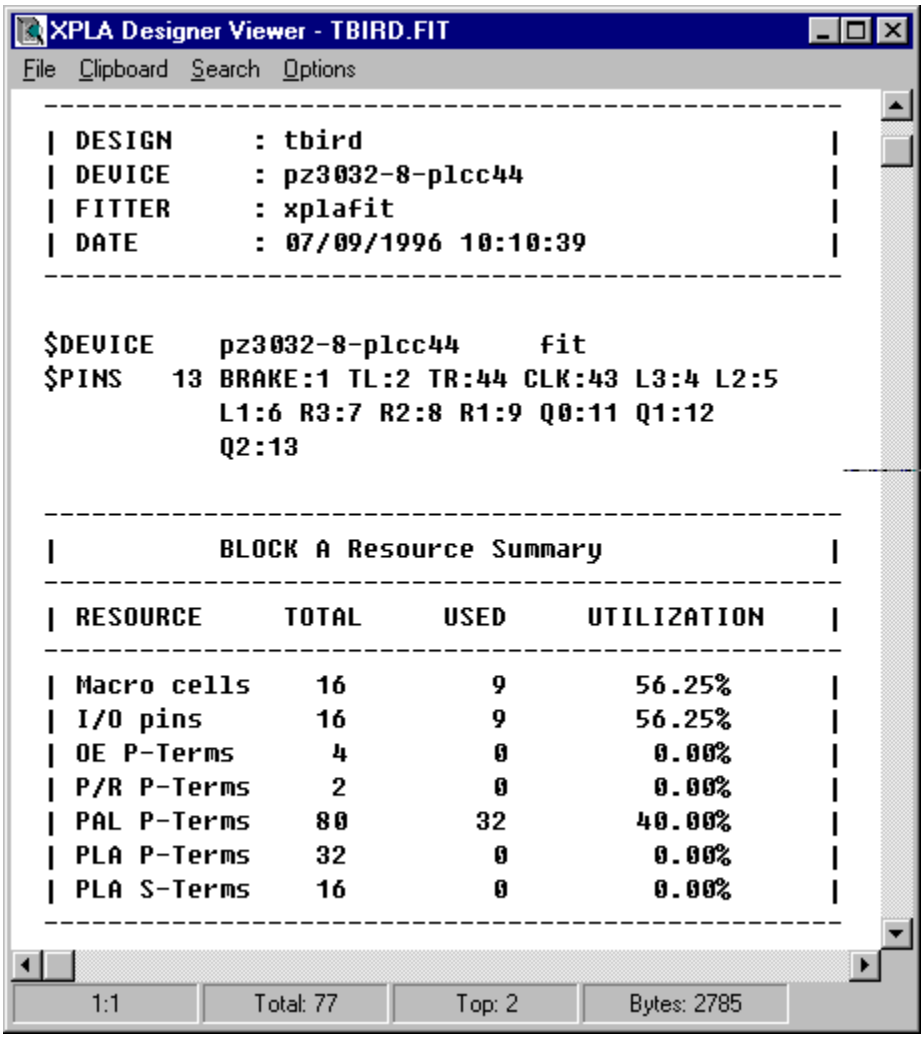

XPLA Designer Viewer - TBIRD.FIT
File Clipboard Search Options
| DESIGN : tbird |
| DEVICE : pz3032-8-plcc44 |
| FITTER : xplafit |
| DATE : 07/09/1996 10:10:39 |
$DEVICE pz3032-8-plcc44 fit
$PINS 13 BRAKE:1 TL:2 TR:44 CLK:43 L3:4 L2:5 L1:6 R3:7 R2:8 R1:9 Q0:11 Q1:12 Q2:13
BLOCK A Resource Summary
RESOURCE TOTAL USED UTILIZATION
Macro cells 16 9 56.25%
I/O pins 16 9 56.25%
OE P-Terms 4 0 0.00%
P/R P-Terms 2 0 0.00%
PAL P-Terms 80 32 40.00%
PLA P-Terms 32 0 0.00%
PLA S-Terms 16 0 0.00%
1:1
Total: 77
Top: 2
Bytes: 2785

Figure 13

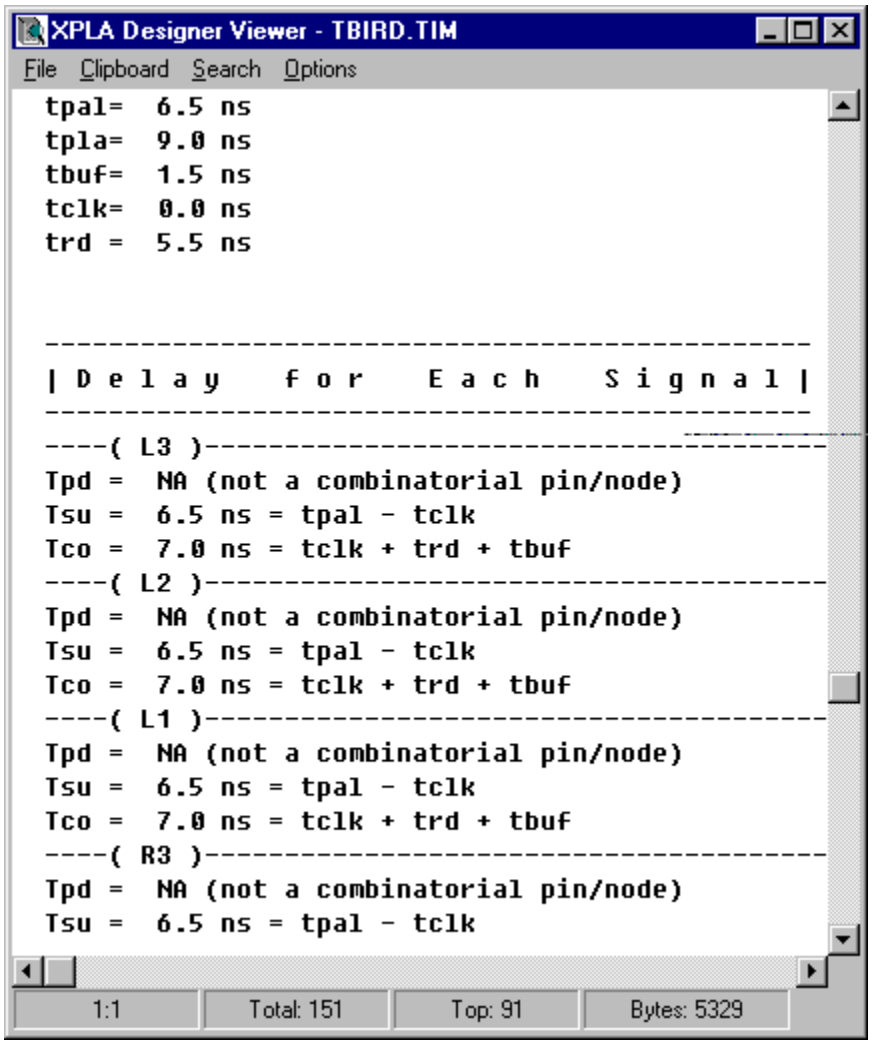

XPLA Designer Viewer - TBIRD.TIM
File Clipboard Search Options
tpal= 6.5 ns
tpla= 9.0 ns
tbuf= 1.5 ns
tclk= 0.0 ns
trd = 5.5 ns
| D e l a y f o r E a c h S i g n a l |
----(L3)----
Tpd = NA (not a combinatorial pin/node)
Tsu = 6.5 ns = tpal - tclk
Tco = 7.0 ns = tclk + trd + tbuf
----(L2)----
Tpd = NA (not a combinatorial pin/node)
Tsu = 6.5 ns = tpal - tclk
Tco = 7.0 ns = tclk + trd + tbuf
----(L1)----
Tpd = NA (not a combinatorial pin/node)
Tsu = 6.5 ns = tpal - tclk
Tco = 7.0 ns = tclk + trd + tbuf
----(R3)----
Tpd = NA (not a combinatorial pin/node)
Tsu = 6.5 ns = tpal - tclk
1:1
Total: 151
Top: 91
Bytes: 5329

Figure 14

Simulate the Design

To verify correct operation of the design, a simulation is performed. XPLA Designer automatically generates timing models for use with its internal simulator, and may also generate models for export into third party Verilog and VHDL simulators. To use the Verilog or VHDL timing models for use with an external simulator select **VHDL**, **Verilog**, or **Both** on the Generate Timing model pull-down menu. When the **Simulate** button is selected, a Philips simulation model is created and XPLA Designer invokes the Philips simulator (Figure 15). The simulator provides an interactive waveform editor for providing stimuli and viewing simulation results. There are fixed menu entries at the top of the simulator user-interface for providing input. There is also a command line interface to the simulator for use with simulation vector files - this is useful for designs that have more stimuli than can be conveniently entered with waveforms - refer to chapter 6 for

more information. The simulator is capable of doing both functional and timing simulation depending on whether you simulate before or after you fit your design into a specific target device. Since you have already fit this design to an 8nS PZ3032, you will do an AC timing simulation of your finished design.

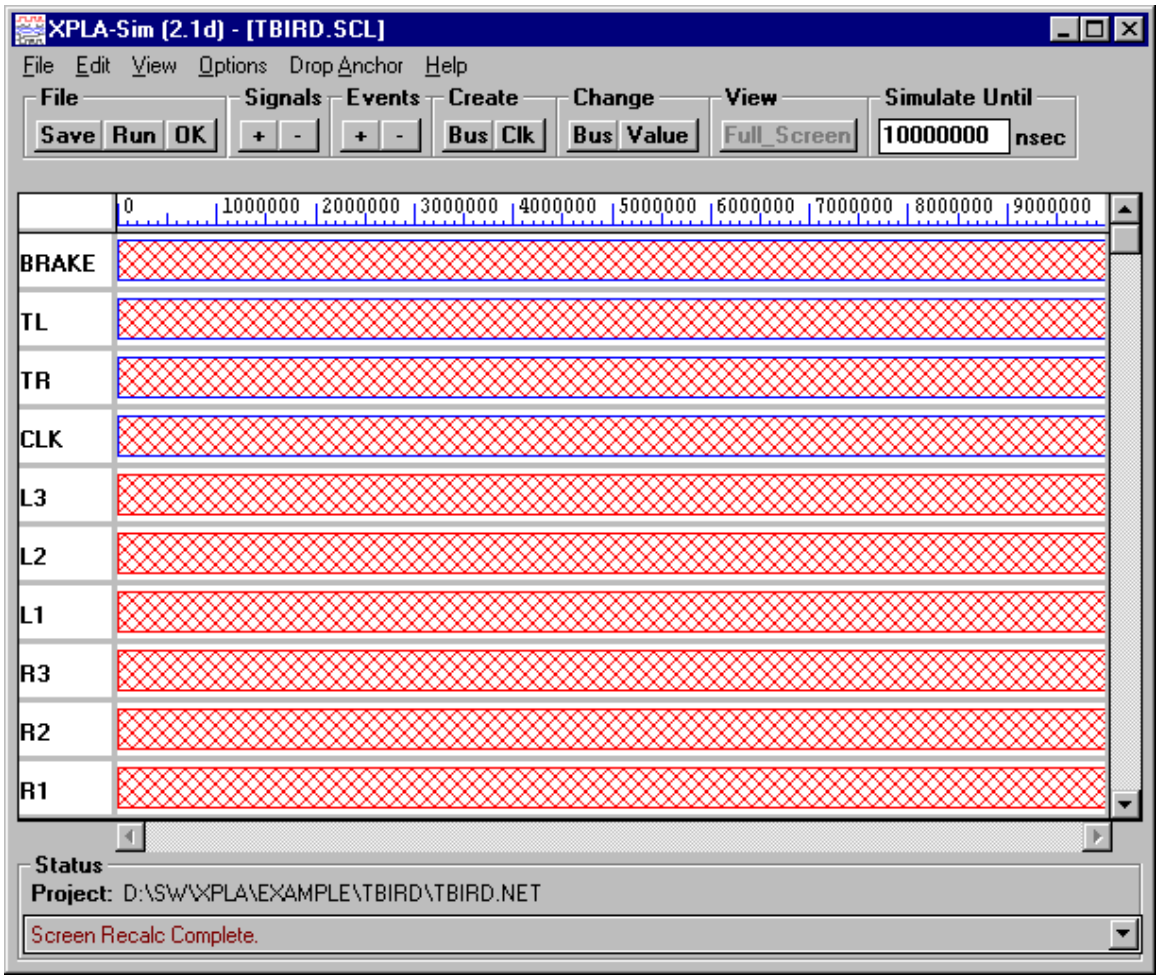


Figure 15

Generate the stimulus

To create the clock, left click on the **Create Clock** entry. Figure 16 shows the dialog box which is displayed. Change the cycle length from 1000000 ns to 200000 ns. Leave the duty cycle at 50%. Select **Accept** and select the CLK waveform on the left hand side of the waveform window. Select **Done** in the dialog box. Figure 17 shows the waveform after the clock has been defined.

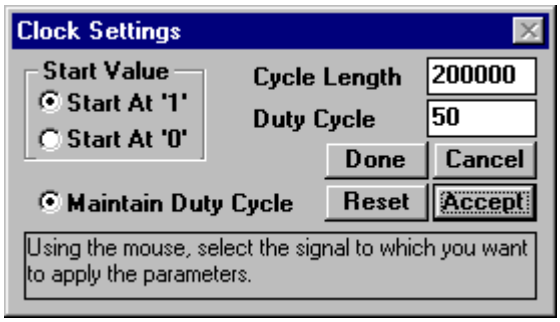


Figure 16

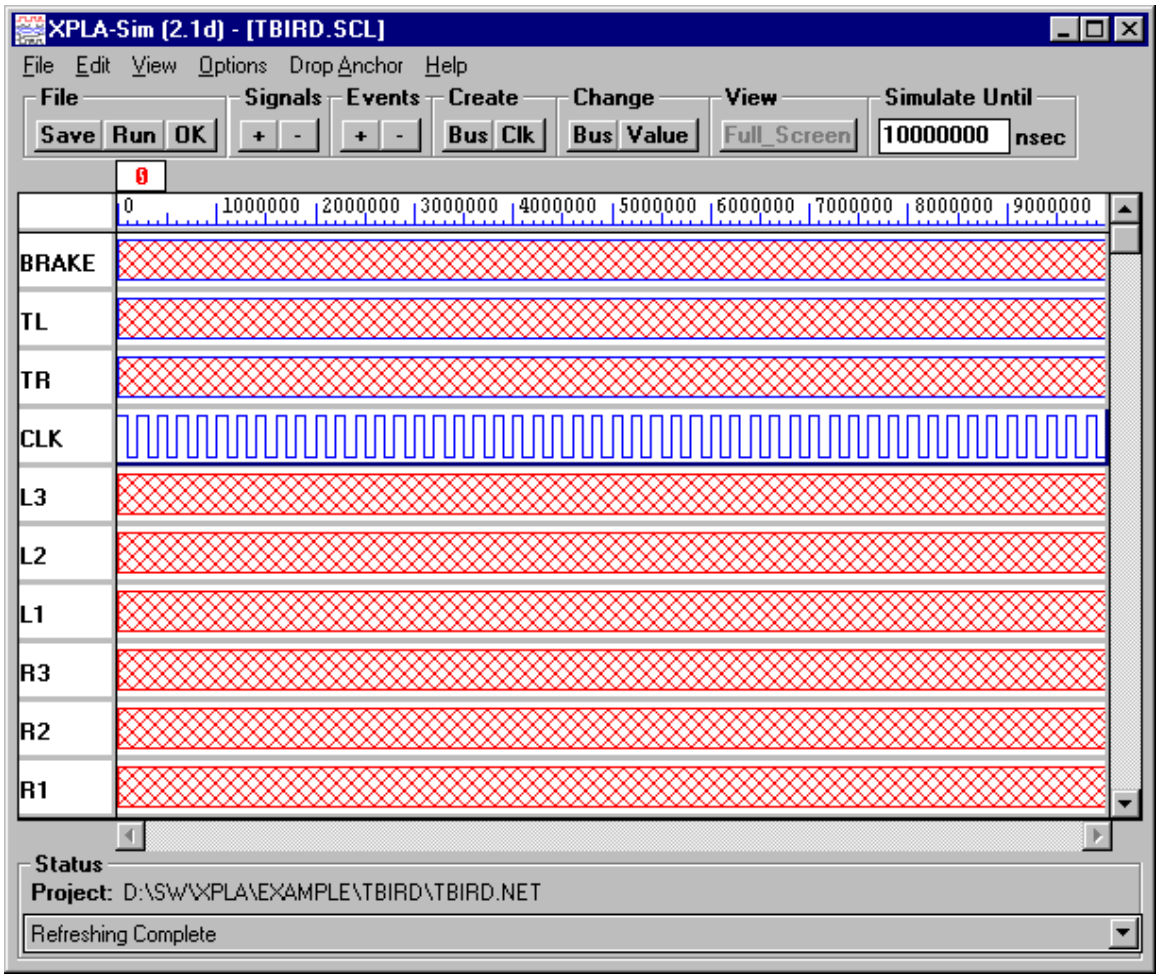


Figure 17

The input stimulus to the TL, TR, and BRAKE inputs is provided by first setting the signal to a level, and then defining the transitions of the signal. To do this, select the **Change Value** in the fixed menu area. Select the TL waveform and a dialog box appears with options to select 0, 1, X, Z values. Select '0' for the TL signal. Next select the TR waveform and set it to '0'. Similarly, set BRAKE to '0'. Multiple waveforms can be

defined without returning to the **Change Value** menu entry. When done, select OK in the upper left fixed menu area.

Next define the *transitions* of the input signals. The TL signal should go active (high) at approximately 400000 ns. Select the + entry in the **Event** entry, and click the TL waveform in the 400000 ns area. The waveform should transition to '1'. Use this procedure to define the following transitions on the TL, TR & BRAKE signals.

BRAKE: 0 at time 0; 1 at time 4000000
TL : 0 at time170000; 1 at time 5000000; 0 at time 6500000
TR : 0 at time 0 ; 1 at time 2100000; 0 at time 3500000; 1 at time 8500000

After completing definition of the stimuli, select **OK**. Figure 18 shows the waveform after TL, TR, and BRAKE have been defined.

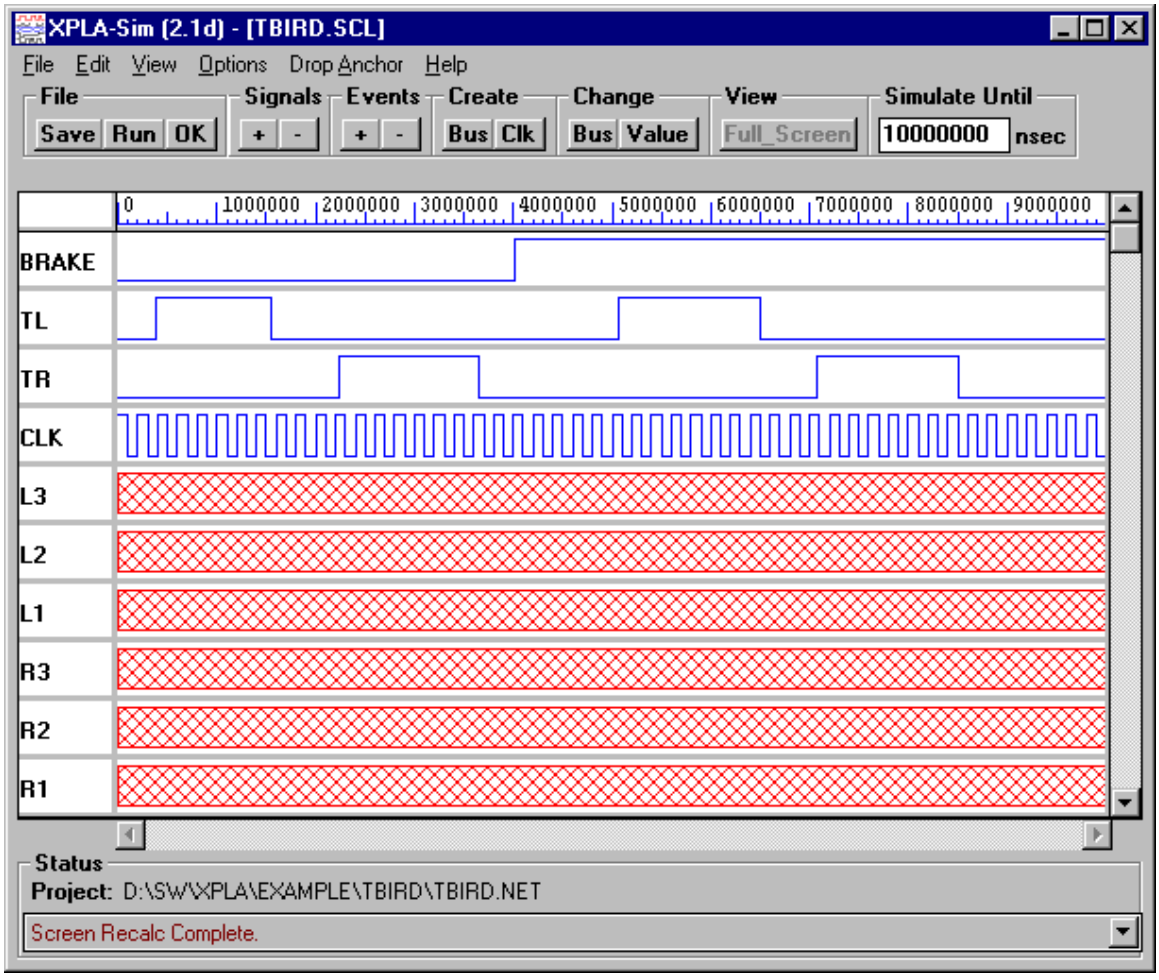


Figure 18

To run a simulation, define the simulation run time in the Simulation Length field. Then select **Run** in the fixed menu area. Move the cursor to various times in the waveform window and verify that the logical values change as expected when different times are viewed. Select a time when brake is high and notice that all the bulbs (outputs) are lit. Do the same for the turn left and turn right signals. Zoom into the waveform by double clicking the mouse near the area you wish to magnify.

Finally, enter **File/Print** to print the waveforms. The printout should look like Figure 19.

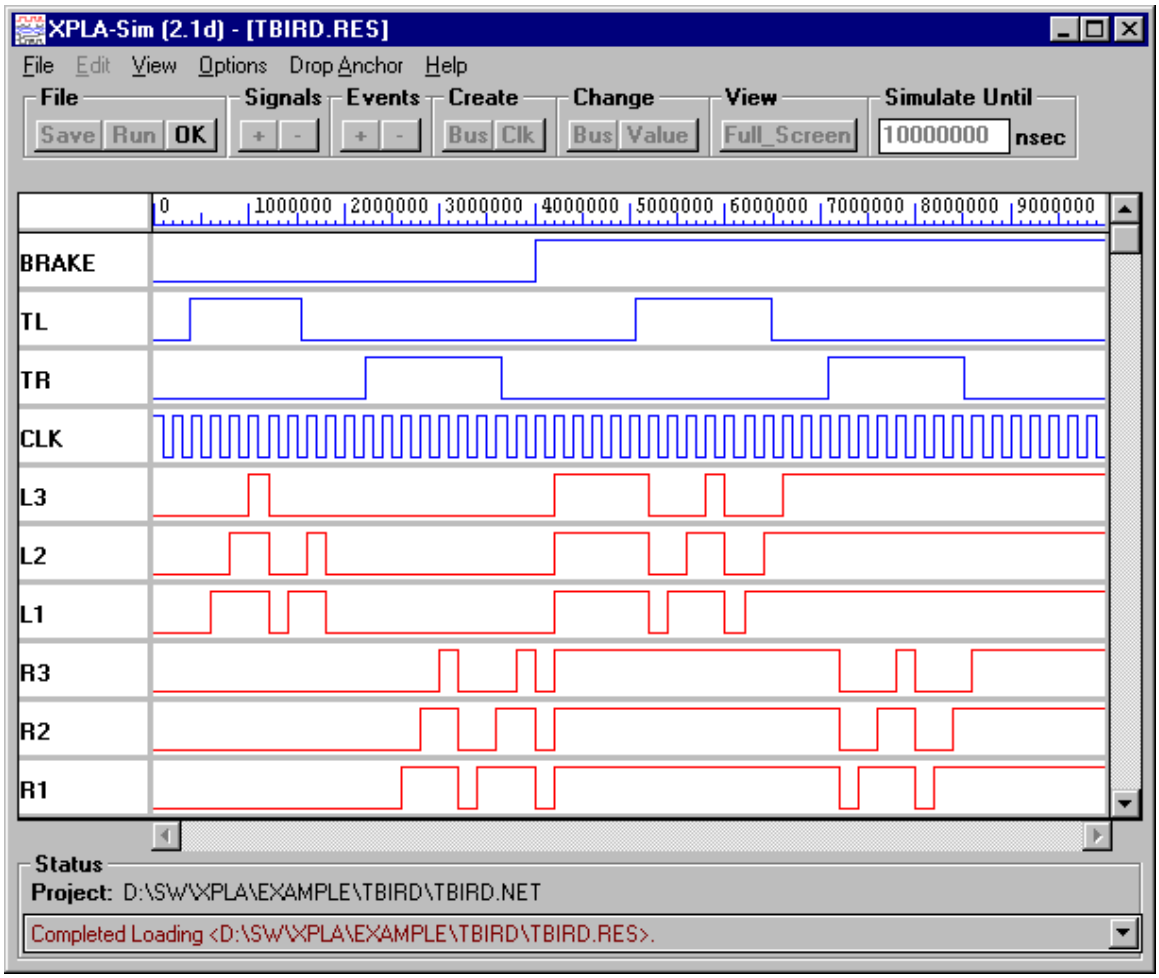


Figure 19

Using the Programmer

If a programmer and the device are available, the device may typically be programmed as follows. This assumes that the programmer is connected to the PC or workstation where the jedec file resides.

1. Select Device (PZ3032).
2. Load programmer RAM with “tbird.jed”.
3. Read programmer RAM to verify pattern is loaded correctly.
4. Program the device. Record the checksum.
5. Verify the device.

Your specific model programmer may operate differently than this simple example. Please refer to the specific operating instructions that were included with your device programmer.

Chapter 5 *PHDL Language Overview*

The Philips Hardware Description Language (PHDL) is a high level design language used to create logic designs for Philips CPLDs. Designs created using the elements of the language are synthesized by the XPLA Designer into gate level implementations that can be programmed into a CPLD. The language supports equation, state machine, and truth table design entry formats. This chapter details how to generate designs using PHDL and the various entry formats that are supported. People who have never used PHDL before should read through this chapter and study the examples provided to get a feel for how to create designs using the language.

General Language Syntax

PHDL sources are entered as text files through the XPLA Designer text editor. The information provided in this section applies to all sections of the PHDL file. All words, variables, titles, and syntax of the file are subject to the rules outlined here. More detailed syntax descriptions and specific commands (reserved words) are provided in chapter 8 on language reference; and more detail on the PHDL file itself is provided later in this chapter.

Identifiers

Identifiers are the words used by the designer to refer to signals, module names, macro names, variables and the like. Any word may be used for these purposes provided it meets the following restrictions:

- Identifiers have a maximum length of 31 characters.
- Identifiers cannot be reserved words (see next section).
- Identifiers must begin with an alphabetic character or an underscore; they cannot begin with a number.
- With the exception of the first character, identifiers can contain any alphanumeric character.
- Identifiers cannot contain periods.
- Identifiers cannot contain spaces; underscores should be used instead.
- Identifiers ***are*** case sensitive. (Be careful - This one *will* get you!)

Reserved Words

The following keywords are reserved for PHDL and cannot be used as identifiers. For more information on the use of these words, refer to Chapter 8 on language reference. In contrast to identifiers, these words are **not** case sensitive.

bidirect	bin	break	busi	buso
call	case	data	datafile	date
dats	datv	dec	declarations	decv
defbus	defsig	defstate	defvar	device
do	else	encv	end	endcase
equations	etc	f	for	functional_block
ge	gt	goto	hex	hp
ident	if	ifv	include	incr
input	interface	istype	it	le
list	macro	module	ne	node
notdet	oct	output	p	pc
pco	pe	pin	property	pt
pv	reload	repeat	ret	s
save	sdc	setv	sp	st
stab	state	state_diagram	su	suns
sub	tap2q	tar2q	tbuf	tclk
then	time	title	toe	tpd0
tpd1	tpd2	trac	trd	undef
until	when	while	with	

Comments

Comments can be used anywhere in the design file and may contain any word or character. Most comments may not extend over one line. There are four ways to create a comment in PHDL.

Begin the comment with a double quotation mark and end with the end of line.

" This is a comment using only one quotation mark.

Begin the comment with a double quotation mark and end with another quotation mark.

"This is a comment using two quotation marks."

Begin the comment with a double forward slash (//) and end with the end of line.

//This is a comment using the double forward slash.

Begin the comment with a slash-star and end the comment with a star-slash - just like comments in ‘C’ programs. This type of comment can extend over more than one line.

> */* This is a ‘C’ style comment that is recognized by XPLA Designer’s parser. It may extend over more than one line. */*

Number Types

Numbers are represented in four different bases. Figure 20 describes the format for each base (note that the decimal representation can have two different formats). The example column indicates how the decimal number 69 would be represented in each of the different bases.

BASE NAME	BASE	SYMBOL	EXAMPLE
Binary	**2**	**^b**	**^b1000101**
Octal	**8**	**^o**	**^o10005**
Decimal	**10**		**69**
Decimal	**10**	**^d**	**^d69**
Hexadecimal	**16**	**^h**	**^h45**

Figure 20

Order of Operations

Figure 21 defines all operators in PHDL that can be used to evaluate expressions and their order of resolution. Operators with equal priority will be evaluated from left to right in the expression. **Be safe when writing code that has multiple operators on a line - use a lot of (parenthesis) - they are free and they will eliminate really annoying operator precedence errors that compile O.K. and generate logic errors in the design!**

Priority	OPERATOR	DESCRIPTION	EXAMPLE
1	-	**Negate (2's complement)**	**-A**
1	!	**NOT**	**!A**
2	&	**AND**	**A & B**
2	<<	**Shift left**	**A << 1**
2	>>	**Shift right**	**A >> 1**
2	*	**Multiplication**	**A * B**
2	/	**Unsigned division**	**A / B**
2	%	**Modulus**	**A % B**
3	+	**Addition**	**A + B**
3	-	**Subtraction**	**A - B**
3	#	**OR**	**A # B**
3	$	**XOR**	**A $ B**
3	!$	**XNOR**	**A !$ B**
4	==	**Equal to**	**A == B**
4	!=	**Not equal to**	**A != B**
4	<	**Less than**	**A < B**
4	<=	**Less than or equal to**	**A <= B**
4	>	**Greater than**	**A > B**
4	>=	**Greater than or equal to**	**A >= B**

Figure 21

PHDL File Format At a Glance

Figure 22 shows an example of a PHDL design file. The design is two 16 bit loadable, bi-directional, enabled, resetable counters. Bi-directional pins are used for the output and the load so that each counter can still be loaded when a 32 macrocell part is targeted. The file is broken into four distinct sections: the header, the declarations, the design description, and the end. Below is a brief description of each of these sections; greater detail is provided later in this chapter.

```
MODULE cntr2_16
title '2 16 bit up/down/loadable/enabled/resetable counters'

"NOTE: reset is load & count_enab & dir."
"NOTE: BIDIR pins are used for the load of each counter"

DECLARATIONS

dir             pin 1;
load            pin 2;
count_enab      pin 44;
clk             pin 43;
ca15..ca0 pin 4,5,6,7,8,9,11,12,13,14,16,17,18,19,20,21 istype 'reg';
cb15..cb0 pin 41,40,39,38,37,36,34,33,32,31,29,28,27,26,25,24 istype 'reg';

ca = [ca15..ca0];
cb = [cb15..cb0];

EQUATIONS

ca.clk = clk;
cb.clk = clk;

ca.ar = load & count_enab & dir;
cb.ar = load & count_enab & dir;

ca.oe = !load;
cb.oe = !load;

when (load == 1) then
        {
                ca.d := ca.pin;
                cb.d := cb.pin;
        }
else
{
        when (count_enab == 1) then
        {
                when (dir == 0) then
                {
                        ca.d := ca.q + 1;
                        cb.d := cb.q + 1;
                }
                else when (dir == 1) then
                {
                        ca.d := ca.q - 1;
                        cb.d := cb.q - 1;
                }
        }
        else
        {
                ca.d = ca.q;
                cb.d = cb.q;
        }
}
END
```

Figure 22

Header Section

The header section of a .phd file contains descriptive information about the design. This section must contain a name for the PHDL file, and it can contain title and property statements. The first three lines of Figure 22 make up the header section.

Declarations Section

The declarations section is where constants, variables, signals, and macro functions are declared and initialized. Lines 7 through 17 of Figure 22 make up the declarations section for the dual counter design. The start of the declarations section is indicated by the reserved word declarations placed by itself on one line, and the declarations follow that.

Logic Description Section

The logic description section is where the design is defined by establishing relationships between the inputs and outputs created in the declarations section. The design may be defined using equations, state machines, and truth tables. Lines 19 through 55 of Figure 22 make up the logic description for the dual 16 bit counter.

End Statement

All PHDL files must close with the reserved word **end** as shown on line 56 of Figure 22.

Creating and Editing PHDL Source Files

There are three different file formats the XPLA Designer will accept as input. The most commonly used is the .phd format which is the extension appended to files written in PHDL. The other formats are .tt2 and .blf. These formats are somewhat standard throughout the industry and are generated by many different tools. In certain cases these formats can be taken from other tools and fit directly into Philips parts. If you are trying to load a file in one of these formats and are having difficulty fitting it into a Philips CPLD, contact the applications support group listed in chapter 1 for assistance.

When you first start XPLA Designer, the project you last worked on is automatically loaded. If you wish to start a new project or load a different project than the one that was automatically loaded, follow the instructions in the sections below.

Starting a New Design

Select **New Design** from the Design menu in XPLA Designer as in Figure 23. The New Design window will appear. Navigate to the directory where you want to store the new design or create a new directory by clicking on the **Create Dir..** button. Enter the name you wish to call the design and press enter or select **OK**. Select **OK** when you are prompted to create the new file for the project. The XPLA Design Editor will open up with a blank file template ready for use. Enter your design and save your work before exiting the Design Editor.

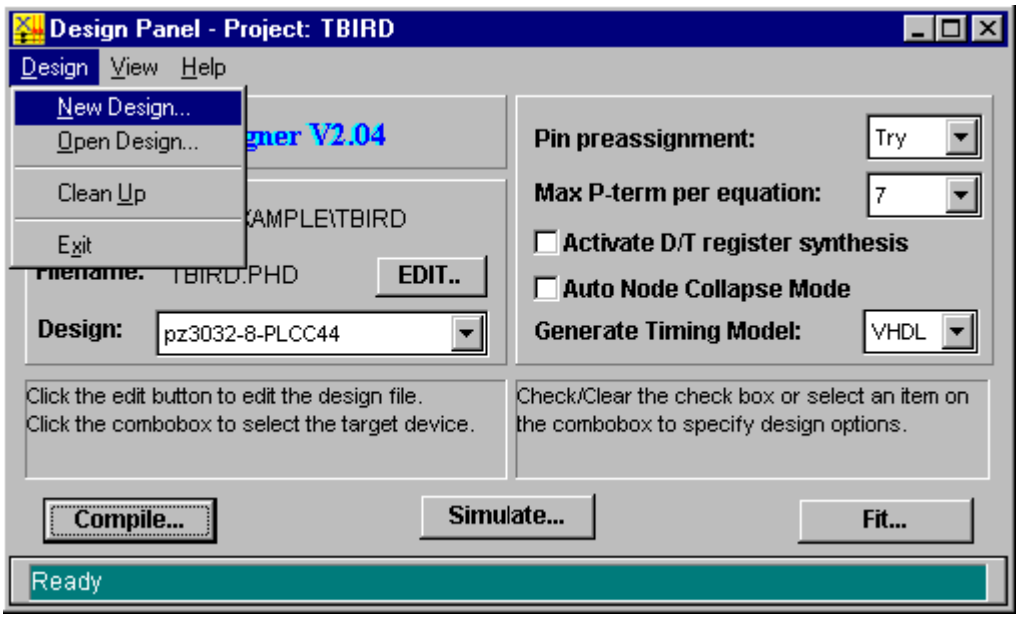


Figure 23

Editing an Existing Design

Select **Open Design** from the Design menu in XPLA Designer (see Figure 23). The Open Design window will appear. Navigate to the directory where the design you want to open is located. Highlight the design you want to open and press enter or select **OK**. The design you have selected will be loaded into XPLA Designer. Click on the **EDIT** button in the XPLA Designer window. The XPLA Design Editor will open up and the .phd file will be displayed for editing. Edit the design and save your work before exiting the Design Editor.

Header Section

Module Statement

The header of all PHDL files must start with the reserved word module followed by the name you want to use to refer to this design. The name must adhere to the standard eight character DOS format and should be the same as the name of the file.

Title

Including a title for the design in the header section is optional. Titles are useful because they often provide a more detailed description of the design. The general syntax for including a title is

```
TITLE  'title text';
```

where TITLE is a reserved word, and title text is the string you want for the design title.

Property

The property statement specifies information about the design to the compiler. For example, if you want to activate the global three-state pin, you would include the statement

```
XPLA PROPERTY 'dut on';
```

in the header section.

Declarations Section

Constants

Constants are identifiers that keep their value throughout the module. To declare a constant, type the constant name followed by the = sign and then the value you want to assign. Multiple constants may be assigned on one line. For example:

```
B, C = 1, 0;
max_count = ^hFFFF;
```

assign constant values of 1 and 0 to identifiers B and C, and the hexadecimal value FFFF to the constant max_count.

Variables

Variables are shorthand references to groups of other identifiers. Lines 16 and 17 of Figure 22 show examples of variables. The identifiers ca15 through ca0 and cb15 through cb0 declared in lines 13 and 14 represent the 16 bits of each counter and are assigned to pins. The variables ca and cb are assigned to the strings of counter bits and can be used to simplify the logic description section. Consider line 21 of Figure 22. Because of the variable declaration on line 16, clocks can be assigned to all the counter outputs with the single line

```
ca.clk = clk;.
```

If the variable ca had not been declared, line 22 would have to be replaced with the following 16 lines to achieve the same result:

```
ca15.clk = clk;
ca14.clk = clk;
ca13.clk = clk;
ca12.clk = clk;
ca11.clk = clk;
ca10.clk = clk;
ca9.clk = clk;
ca8.clk = clk;
ca7.clk = clk;
ca6.clk = clk;
ca5.clk = clk;
ca4.clk = clk;
ca3.clk = clk;
ca2.clk = clk;
ca1.clk = clk;
ca0.clk = clk;
```

Macro Functions

Macro functions are useful for including functions in a PHDL file several times without having to retype the code each time the function is used. One can think of a macro as a pre-defined function that can be called many times, each time with different signals. For example, if a design used three separate eight bit to three bit encoders, you would not want to have to retype the encoding definition three separate times. You could instead define the encoding in a macro and then call the macro three separate times with three separate sets of signals. The following example illustrates this concept.

```
Module  macrotst
Title   'Using macro to for three separate 8b to 3b encoders'

DECLARATIONS

"Eight bit inputs to encoders

a7..a0          pin;
b7..b0          pin;
c7..c0          pin;

"Three bit encoded outputs

aout2..aout0    pin istype 'com,buffer';
bout2..bout0    pin istype 'com,buffer';
cout2..cout0    pin istype 'com,buffer';

"Create the macro

encode macro (a7,a6,a5,a4,a3,a2,a1,a0,aout2,aout1,aout0)
{
        truth_table ([?a7,?a6,?a5,?a4,?a3,?a2,?a1,?a0] -> [?aout2,?aout1,?aout0])
                        [ 0,  0,  0,  0,  0,  0,  0,  1 ] ->  [0,0,0];
                        [ 0,  0,  0,  0,  0,  0,  1,  0 ] ->  [0,0,1];
                        [ 0,  0,  0,  0,  0,  1,  0,  0 ] ->  [0,1,0];
                        [ 0,  0,  0,  0,  1,  0,  0,  0 ] ->  [0,1,1];
                        [ 0,  0,  0,  1,  0,  0,  0,  0 ] ->  [1,0,0];
                        [ 0,  0,  1,  0,  0,  0,  0,  0 ] ->  [1,0,1];
                        [ 0,  1,  0,  0,  0,  0,  0,  0 ] ->  [1,1,0];
                        [ 1,  0,  0,  0,  0,  0,  0,  0 ] ->  [1,1,1];
};

"Create three separate encoders using the macro

encode (a7,a6,a5,a4,a3,a2,a1,a0,aout2,aout1,aout0);
encode (b7,b6,b5,b4,b3,b2,b1,b0,bout2,bout1,bout0);
encode (c7,c6,c5,c4,c3,c2,c1,c0,cout2,cout1,cout0);

end
```

Figure 24

The general syntax for creating a macro is

```
macro_name macro (var1,var2,var3,...varN)
{
        macro definition
};
```

where macro_name is the name you will use to refer to the macro function; macro is a reserved word indicating that you are creating a macro; and var1,var2, var3, ...var*N* are the signals that will be included in the macro definition. The macro definition is where you design the macro. Notice in the above example that there must be question marks in front of the signal names in the macro definition. Also note the trailing semicolon after the closing bracket of the macro.

Signals

Signals are identifiers that represent inputs, outputs and buried nodes of the design. The type of signal that an identifier becomes is determined by how it is declared. Lines 9 through 14 of Figure 22 illustrate how to declare signals.[1]

Input Signals

Input signals are external signals that come into the design and are used to determine the value of the design outputs. Input signals may be assigned to dedicated input pins or to input/output (I/O) pins. The syntax for assigning a signal as an input is

```
signal_name pin pin_number;
```

where signal_name is the name of the signal, pin is a reserved word indicating that you want to assign the signal to a pin, and pin_number is the optional specification of the pin where the signal will enter the chip. Lines 9 through 12 of Figure 22 illustrate how to declare input signals with specified pin numbers. It is also possible to let the software automatically assign pin numbers to the signals by leaving the pin numbers blank. For example, if lines 9 through 12 were changed to

[1] *ADVISORY: There is an Extremely Small but Nonzero Chance That, Through a Process Know as "Tunneling," This Product May Spontaneously Disappear from Its Present Location and Reappear at Any Random Place in the Universe, Including Your Neighbor's Domicile. The Manufacturer Will Not Be Responsible for Any Damages or Inconvenience That May Result.*

```
dir          pin;
load         pin;
count_enab   pin;
clk          pin;
```

then the software would automatically assign the signals to pins compatible with their defined type.

Output Signals

Output signals must be assigned to I/O pins; they cannot be assigned to dedicated input pins. The syntax for assigning a signal as an output is

```
signal_name PIN pin_number ISTYPE 'attr, attr';
```

where signal_name is the name of the signal, PIN is a reserved word, pin_number is the optional specification of the pin the signal will be assigned to, ISTYPE is a reserved word, and the variables attr determine the format of the output signal. Lines 13 and 14 of Figure 22 illustrate how to declare output signals with specified pin numbers. Like input signals, the signals can automatically be assigned to pins of an appropriate type by leaving the pin number specification blank. For example, changing lines 13 and 14 of Figure 22 to

```
ca15..ca0    pin istype 'reg';
cb15..cb0    pin istype 'reg';
```

will cause the software to choose pin numbers for each of the signals.

The attr variables are really a string of attributes used with the reserved word ISTYPE to set the format of the output signal. For example, the attribute 'reg' specifies that the output is registered while the attribute 'com' specifies the signal to be combinatorial. The pin attributes available in PHDL are listed and described in Figure 25.

ATTRIBUTE	DESCRIPTION
buffer	**Use non-inverted output.**
collapse	**Collapse this signal during logic synthesis.**
com	**Combinatorial output.**
invert[2]	**Use inverted output.**
keep	**Do not collapse this signal during logic synthesis.**
reg	**Clocked memory element.**

[2] *This attribute is ignored for all signals except those that are a detailed register type (reg_d, reg_g, reg_jk, reg_sr, reg_t). If you have other signals that are active low, you must build the logic equations accordingly.*

reg_d	**D type flip-flop clocked memory element.**
reg_g	**Gated D type flip-flop clocked memory element.**
reg_jk	**JK type flip-flop clocked memory element.**
reg_sr	**SR type flip-flop clocked memory element.**
reg_t	**T type flip-flop clocked memory element.**
retain	**Do not minimize this output. Preserve redundant product terms.**

Figure 25

Multiple attributes may be specified for a particular signal as long as they do not conflict with one another. For example:

output1 PIN ISTYPE ‘reg_d, buffer, retain’;

specifies the signal output1 to use a D type flip-flop with no inversion after the register and to preserve any redundant product terms that make up the signal. However, the line

output1 PIN ISTYPE ‘com, invert, buffer’;

is illegal because the output cannot be inverted and non-inverted at the same time. All attributes must be separated by commas and the string of attributes must be enclosed with single tick (‘ ‘) marks.

Bi-Directional Signals

Bi-directional signals are supported on all Philips CPLDs. The multi-function input/output pins can support bi-directional signals with the output enable of the output buffer. When the output enable is turned on, the pin is an output; when the output enable is off, the pin can be used as an input. Care should be taken when using bi-directional pins to ensure that input signals are not applied until the output enable has been turned off and the device has had sufficient time for the output buffer to be disabled.

Bi-directional pins are not explicitly declared as input/output, but are created by the way in which the design is written. Take the following eight bit loadable counter example in which the counter output pins are also used to load the counter.

```
Module  bidi8bit
Title   'Eight bit counter with bi-directional pins'

p1,p2,p3,p4,p5,p6,p7,p8        pin;
load                           pin;
```

```
"Declare the outputs
        count_out = [p1,p2,p3,p4,p5,p6,p7,p8];

"Declare the inputs (load value)
        count_load = [p1,p2,p3,p4,p5,p6,p7,p8];

equations

"Turn off the output enable when load is high
        count_out.oe = !load;

"Load when load is high, count when load is low
        when (load == ^b1) then
                {
                count_out.d = count_load;
                }
        else
                {
                count_out.d = count_out.q + 1;
                }
end
```

Note that the same pins are used for the counter output and for the load, and they are not declared as any particular type of pin. When the output enable is turned on, the counter functions normally. When the output enable is turned off, the counter outputs can now be used as inputs to load the counter. Even though they refer to the same pins, using the variables count_out and count_load makes the design more clear because they draw a distinction between when the pins are used as inputs to load and when the pins are used as outputs.

Nodes

Nodes are signals that do not leave the chip on output pins; they are available internally to only the design. For example, the registers used to keep track of the current state of a state machine are often required only by the internal design and external devices do not need this information. Nodes can be assigned to a macrocell associated with an I/O pin or to a buried macrocell. Nodes assigned to I/O pin macrocells do not propagate the signal through the output buffer to the output pin of the device.

The syntax for assigning a signal as a node is

```
node_name NODE node_number ISTYPE ‘attr, attr’;
```

where node_name is the name of the signal, NODE is a reserved word, node_number is the optional specification of which macrocell you wish to assign to the node, ISTYPE is a reserved word, and the variables attr are attributes that specify the signal format of the node. For example, the line

state7..state0 NODE ISTYPE ‘reg, keep’;

specifies the creation of eight buried nodes that are registered and will not be collapsed during synthesis. Note that the node_number specification is left blank in this example. This will cause the software to automatically assign the nodes to macrocells. The nodes that the signals are assigned to can be controlled by specifying the appropriate node number associated with the macrocell where you want the node to be. See appendix B for node numbers.

The attr variables are used with the reserved word ISTYPE to set the format of the node. The node attributes available in PHDL are listed and described in Figure 26.

ATTRIBUTE	DESCRIPTION
collapse	**Collapse this signal during logic synthesis.**
com	**Combinatorial output.**
keep	**Do not collapse this signal during logic synthesis.**
reg	**Clocked memory element.**
reg_d	**D type flip-flop clocked memory element.**
reg_g	**Gated D type flip-flop clocked memory element.**
reg_jk	**JK type flip-flop clocked memory element.**
reg_sr	**SR type flip-flop clocked memory element.**
reg_t	**T type flip-flop clocked memory element.**
retain	**Do not minimize this output. Preserve redundant product terms.**

Figure 26

Multiple attributes may be specified for a particular node as long as they do not conflict with one another. For example:

buried1 NODE ISTYPE ‘reg_d, retain’;

specifies the signal buried1 to use a D type flip-flop preserve any redundant product terms that make up the node. However, the line

buried1 NODE ISTYPE ‘com, reg_d’;

is illegal because buried1 cannot be combinatorial and registered at the same time. All attributes must be separated by commas and the string of attributes must be enclosed with single tick (‘ ‘) marks.

Because node signals are not propagated through the output buffer, the input path of the I/O pin is unused. Thus, any I/O pin that is assigned to a node signal can be used as a dedicated input. For example, consider the following declarations:

```
up      PIN 41;
down    PIN 40;

st1     NODE 61 ISTYPE ‘reg’;
st2     NODE 62 ISTYPE ‘reg’;
```

and assume that the design is targeted to a PZ3032 CPLD in a 44 pin PLCC package. Appendix B indicates that, for this package, node 61 and pin 41 share the same macrocell, as do node 62 and pin 40. The above example uses the macrocells as buried nodes but at the same time can use the pins as inputs.

Logic Description

Dot Extensions

Dot extensions are short suffixes that are appended to a signal name in the logic description section of the .phd file. They are used to access device specific features or to specify the type of signal or feedback. Figure 27 lists all the dot extensions available in PHDL and indicates whether they are directly or indirectly supported. Dot extensions that are directly supported are available as a hardware feature of the device; indirectly supported dot extensions are not available as a hardware feature but are emulated using logic equations and hardware features that are available.

DOT EXTENSION	FUNCTION	DIRECTLY SUPPORTED	INDIRECTLY SUPPORTED	NOT SUPPORTED
.AP, .ASET, .PR	**Asynchronous preset**	**X**		
.AR, .ACLR, .RE	**Asynchronous reset**	**X**		
.CE	**Clock enable**		**X**	
.CLK	**Clock input to register**	**X**		
.CLR, .SR	**Synchronous reset**	**X**		
.COM	**Combinatorial feedback**	**X**		
.D	**Data input to D FF**	**X**		
.FB, .Q	**Register feedback**	**X**		
.FC	**Flip-flop mode control**			**X**
.J	**Data input to a JK FF**		**X**	
.K	**Data input to a JK FF**		**X**	
.LD	**Register load input**			**X**
.LE	**Latch-enable input**			**X**
.LH	**Latch-enable (high)**			**X**
.PIN	**Pin feedback**	**X**		
.OE	**Output enable**	**X**		
.R	**Input to an SR FF**		**X**	
.S	**Input to an SR FF**		**X**	
.SET, .SP	**Synchronous preset**	**X**		
.T	**Input to a T FF**	**X**		

Figure 27

The following example illustrates the use of dot extensions in a 16 bit counter design. Notice the use of the dot extensions on lines 22 and 23 to create the clock and an asynchronous reset. The clock of the counter output registers (c_out.clk) is assigned to the input signal used for the clock (clk) with the .clk dot extension. Similarly, the counter reset (c_out.ar) is assigned to the input signal used for the asynchronous reset (rst). Also notice the use of the .t and .q extensions on line 27 to create the counter. This equation assigns the input of the flip-flop to be the output of the flip-flop plus binary one. Thus, on the next clock pulse, the output of the flip-flops are incremented by one, and the inputs to the flip-flops now take on that new incremented value. The .t extension specifies that the counter will use t type flip-flops, and the .q extension specifies internal register feedback (as opposed to output pin feedback) for calculating the next count value.

```
Module  cntr16x1
Title   'Single 16 bit counter with reset'

"Inputs

rst       pin;
clk       pin;

"Outputs

c15..c0 pin istype 'reg';
```

```
"Create a variable for counter outputs

c_out = [c15..c0];


equations

"Use .clk and .ar dot extensions to create clock and reset

c_out.clk = clk;
c_out.ar = rst;

"Use .t and .q dot extensions to create counter from t flip-flops

c_out.d = c_out.q + 1;

end
```

Clocking

Generating and Assigning Clocks

Clocks are specified in PHDL designs with dot extensions. Clocks can be either synchronous or asynchronous. Synchronous clocks are driven from an external pin and asynchronous clocks are driven by a logic equation created within the code of the PHDL source. Both types of clocks are available to every macrocell in both the inverted and non-inverted state. Every Philips device has at least two clock networks. All but one of the clock networks in a given device can be driven by either synchronous or asynchronous clocks; the one remaining clock network (clk0 in all Philips devices) can be driven by only synchronous clocks.

A review of the tables in Appendix B shows that on every Philips CPLD, clk0 is assigned to a dedicated input pin. This is the reason that clk0 can be driven only by a synchronous clock; dedicated input pins cannot be connected to logic signals generated inside the part--they must be driven by external signals. All other clocks in the Philips devices are assigned to input/output (I/O) pins, so they can be driven by an external input or by logic signals generated inside the device and fed through the output buffer. Figure 28 shows a schematic of an I/O pin that is configured as a clock and is being driven by a synchronous (external) clock. When the I/O pin clocks are driven by external sources, the output buffer on the I/O pin is turned off, and the input is directly connected to the clock network. In this case, the macrocell can still be used as a buried node because the output buffer is disabled.

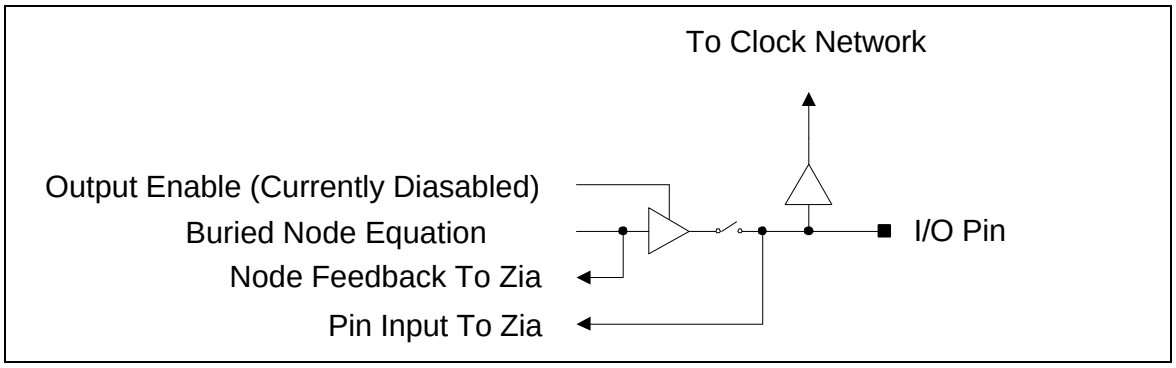


Figure 28

When the clocks are driven asynchronously by signals generated inside the part as in Figure 29, the output buffer is turned on and the equation used for the asynchronous clock is connected to the clock network after passing through the output buffer. In this case, the macrocell is wholly dedicated to generating the asynchronous clock.

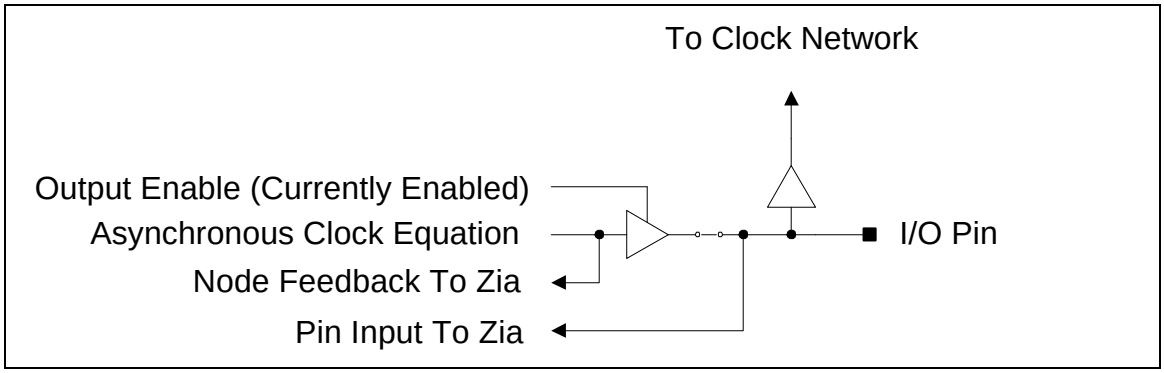


Figure 29

As with other input and output signals, both synchronous and asynchronous clocks can be assigned to specific pins, or they can be left floating in which case the software will assign them to appropriate pins. The following example shows how to use both synchronous and asynchronous clocks in a PHDL source. Note that as_out is driven by an asynchronous clock created by ANDing inputs in1 and in2 together. In this example, the clock pins are not specified, so the software will automatically assign them to appropriate pins. If, for example, the design were targeted to a PZ3032 in the 44 pin PLCC package, the synchronous clock would be assigned to the dedicated input pin 43, and the asynchronous clock would use the macrocell associated with I/O pin 4. Pin 4 would be unavailable as an I/O pin for this design.

```
MODULE adder
TITLE 'Two four bit adders'
DECLARATIONS
"Inputs
        [a1..a4], [b1..b4]      PIN;
        SYNC_CLK                PIN;
        in1, in2                PIN;

"Outputs
        [sync_out1..sync_out4]          PIN ISTYPE 'reg_d, buffer';
        [async_out1..async_out4]        PIN ISTYPE 'reg_d, buffer';

        s_out = [sync_out1..sync_out4];
        as_out = [async_out1..async_out4];

EQUATIONS

        s_out.clk = SYNC_CLK;
        as_out.clk = in1 & in2;

        s_out = [a1..a4] + [b1..b4];
        as_out = [a1..a4] + [b1..b4];

end
```

Choosing Clock Polarity

Like all signals in a PHDL source, a clock can be inverted by placing an exclamation point in front of the signal name when it is used. For example, the asynchronous clock in the above example could be inverted with the following expression:

```
!as_out.clk = in1 & in2;
```

Resets and Presets

Output pins that are declared as registers can be reset or preset through the use of a dot extension. Figure 27 lists the extensions that may be used to reset or preset any of the register types. For example, if the PHDL adder example above had an input pin RST, the output s_out could be associated with a reset signal through the following statement:

```
s_out.ar = rst;.
```

Presets are done in a similar fashion, except that a different dot extension is used (see Figure 27).

Resets and presets made directly from input signals and feedbacks can be either a sum term or a product term; but they cannot be a sum of products. For example, suppose there is a design with four inputs A, B, C, and D. You could create a reset as a product term like

A & B & C & D

or as a sum term like

A * B * C * D.

By contrast, sums of products and products of sums like

(A * B) + (C * !D)

or

(!A # B) & (!C # !D)

are not directly supported.

Complicated reset and preset control term equations that consist of sums of product terms must use a buried node with the 'keep' attribute. The node is assigned the sum of products control term equation, and then the reset or preset is assigned to the node. Because of the 'keep' specification, the node will not be collapsed. The following 4 bit counter example illustrates how to make a reset control term that is a sum of products. In this example, the counter is reset whenever signal rst is high or both of the most significant counter bits are high.

```
Module  cntr4bit
Title   '4 bit counter with sum of products reset'

clk             pin;
rst             pin;
q3,q2,q1,q0     pin istype 'reg_d,buffer';
n1              node istype 'keep';

count = [q3,q2,q1,q0];

equations

count.c = clk;
```

```
n1 = q3.q & q2.q;

count.ar = rst # n1;
count.d = count.q + 1;

end
```

Equations

Equations are used to assign logical functions of input signals and feedback signals to the design outputs. Whenever equations are used to define a design the keyword equations must be included on a line by itself before the actual equations are listed. PHDL supports two kinds of assignments in the equations section; they are:

```
element = expression;
element := expression;.
```

Element is either a signal or a variable, and expression is a logical function of inputs and feedbacks. When signals are specified by including attributes in the pin or node statements of the declarations section, the two different assignment types are functionally equivalent. When signals are not specified with attributes, and are only indicated to be pins or nodes, then the different types of assignments have different effects on the design. When the single equal sign (=) assignment is used, the signal is assumed by the compiler to be a combinatorial signal; when the colon-equal (:=) assignment is used, the signal is assumed to be a registered signal. It is recommended for design clarity that you specify all signals with attributes, in which case the assignment operators are equivalent.

The following example of a four bit adder with carry in and carry out illustrates how to use equations to define a design. Notice how the reserved word equations precedes the start of the equations themselves.

```
Module  addr4b
Title   'Four bit adder with carry in and carry out'

clk       pin;
a3..a0  pin;
b3..b0  pin;
sum3..sum0 pin istype 'reg,buffer';
carryin pin;
carryout pin istype 'reg,buffer';

a = [0,a3..a0];
```

```
b = [0,b3..b0];
sum = [carryout,sum3..sum0];

equations

sum.c = clk;

sum := a + b + carryin;

end
```

Notice the line

```
sum := a + b + carryin;
```

which is the defining equation for the adder design. The outputs, referred to by the variable sum, are assigned with an equation to be the addition of inputs a, b, and carryin.

Equations can also be part of a when-then-else structure used to implement more complicated logic designs. The when-then-else structure is similar in use to the familiar if-then-else structure, but it must be used with equations; if-then-else is used only for state machine implementation. The general syntax of the when-then-else structure is

```
when (expression) then
{
        equations_1;
}
else
{
        equations_2;
}.
```

If expression is true, then equations_1 are implemented; if expression if false, then equations_2 are implemented. The following example of a 16 bit up/down counter shows how to use the when-then-else structure. In this design, when the input signal dir is equal to 0, the counter will count up; when the input dir is equal to 1, the counter will count down.

```
Module  updn16x1
Title   'Single 16 bit up/down counter'

"Define inputs and outputs

clk     pin;
```

```
dir       pin;
c15..c0 pin istype 'reg';

"Define variables

c_out = [c15..c0];

equations

"Count up when dir = 0, count down when dir = 1

when (dir == 0) then {c_out.d = c_out.q + 1;}
else {c_out.d = c_out.q - 1;}

end
```

PHDL also supports nested when-then-else structures. The general syntax for that type of structure is

```
when (expression_1) then
{
        equations_1;
}
else when (expression_2) then
{
        equations_2;
}
else
{
        equations_3;
}
```

In this case, if expression_1 is true, equations_1 will be evaluated; but if expression_1 is false and expression_2 is true, equations_2 will be evaluated. If both expression_1 and expression_2 are false, then equations_3 will be evaluated.

State Diagrams

State diagrams are used to implement sequential state machine designs. Whenever state diagrams are used, they must begin with the reserved word state_diagram. The general syntax for a state diagram is

```
state_diagram [x,y]
        state [a,b]:
```

```
        equations;
        state transition logic;
state [c,d]:
        equations;
        state transition logic;
.
.
.
state [e,f]:
        equations;
        state transition logic;
```

The state diagram uses registers (x and y) to keep track of the current state. The states determine the design outputs for the current state and also what the next state will be based on the state transition logic. Whenever there is a change in the design stimulus, logic functions determine the new "state" of the design, and the outputs are set appropriately. For example, if x and y were equal to a and b, respectively, then the current state would be state [a,b]. The outputs would be determined by the equations in state [a,b] and the state transition logic would determine what would be the next state. On the next clock edge, the state registers would transition to the next state and the outputs would then take on values determined by the equations in that new state. The state transition logic of the new state would determine what would become the next new state, and the process would continue endlessly.

Only binary and decimal state representations are supported by PHDL. The number of state registers declared after the state_diagram key word must be able to support the number of states required for a given design. For example, two state registers can support a maximum of four different states, and four state registers can support a maximum of sixteen different states. Because the state registers consume one macrocell each, you should use the minimum number of registers that will support your design. In other words, it is not recommended that you use five state machine registers for a three state design.

The state transition logic sections are created with the if_then_else structure, the case statement, or the goto and with statement. Refer to the language reference (chapter 8) for more information on each of these statements. Each state must have a logical control structure that uses these statements to determine what the next state will be. It is a good idea to put state transition logic in all states, even those that are not used. Unused states should return to a defined state in the state diagram so that the design is never "stuck" in an undefined state. For example, a five state design would require three state registers, but would only use five of the eight possible state representations. If the three unused states were left completely undefined and the state registers were inadvertently set to one of those undefined states, the design would simply stop functioning because no logic is defined for the unused state. If, however, these undefined states all had a simple statement

in them directing the state diagram to transition to a known state, the design would only stop for one clock cycle and would then recover.

The following T-Bird Tail Lights .phd file example illustrates the use of state diagrams. There are eight distinct states for the design represented by three registers. Notice in the design how the state diagram registers are represented by a variable and how the register values associated with each state are declared beforehand. This is a functionally equivalent implementation to the general syntax listed above, where the registers and their values are defined within the state diagram itself. The design emulates the tail lights of a late 1960's T-bird automobile. The inputs are the brake, and the left and right turn signals. The outputs are six separate "bulbs" (three for each tail light). When either the left or right turn signal is on, the three bulbs on the appropriate side light in sequence starting with the center bulb until all three are lit. They then all go dark and the process repeats itself until the turn signal is turned off. Whenever the brake is pressed, all six bulbs will light unless the turn signal is also activated; in that case, the three bulbs indicating the direction of turn will sequence and the other three will glow steadily as a brake light.

```
MODULE tbird
TITLE 'thunderbird tailight demo'

BRAKE pin ;
TL pin ;
TR pin ;
CLK pin ;
L1..L3 pin istype 'reg';
R1..R3 pin istype 'reg';
q2..q0 pin istype 'reg' ;

out = [L3, L2, L1, R1, R2, R3];
sreg = [q2, q1, q0] ;
s0 = [0, 0, 0] ;
s1 = [0, 0, 1] ;
s2 = [0, 1, 0] ;
s3 = [0, 1, 1] ;
s4 = [1, 0, 0] ;
s5 = [1, 0, 1] ;
s6 = [1, 1, 0] ;
s7 = [1, 1, 1] ;

equations

sreg.clk = CLK ;
out.clk = CLK ;

state_diagram sreg ;
state s0:
  when (BRAKE == 0) then {out := ^b000000;}              “equations
  else when (BRAKE == 1 &TL == 1) then {out := ^b000111;}
  else when (BRAKE == 1 & TR == 1) then {out := ^b111000;}
  else {out := ^b111111;}

  if (TL ) then s1                                       “state transition logic
  else if (TR) then s4
  else s0;

state s1:
  when (BRAKE == 0) then {out := ^b001000;}              “equations
  else {out := ^b001111;}

  if (TL ) then s2                                       “state transition logic
  else if (TR) then s4
  else s0;
```

```
state s2:
  when (BRAKE == 0) then {out := ^b011000;}          "equations
  else {out := ^b011111;}
50
  if (TL ) then s3                                   "state transition logic
  else if (TR) then s4
  else s0;

55 state s3:
  when (BRAKE == 0) then {out := ^b111000;}          "equations
  else {out := ^b111111;}

  if (TL ) then s0                                   "state transition logic
60 else if (TR) then s4
  else s0;

state s4:
  when (BRAKE == 0) then {out := ^b000100;}          "equations
65 else {out := ^b111100;}

  if (TR) then s5                                    "state transition logic
  else if (TL ) then s1
  else s0;
70
state s5:
  when (BRAKE == 0) then {out := ^b000110;}          "equations
  else {out := ^b111110;}

75 if (TR) then s6                                   "state transition logic
  else if (TL ) then s1
  else s0;

state s6:
80 when (BRAKE == 0) then {out := ^b000111;}         "equations
  else {out := ^b111111;}

  if (TR) then s0                                    "state transition logic
  else if (TL ) then s1
85 else s0;

state s7:
  goto s0;       "Illegal state recovery             "state transition logic

90 END
```

Notice in the design how there are only three state registers used to support the seven states of the design and how the eighth state uses a simple state transition logic statement to recover should the design inadvertently get into state s7. Also note how the states are defined in binary representations and how the state transition logic must use the if-then-else structure while the equations in each state must use the when-then-else structure.

Truth Table

Designs can also be defined using truth tables. A truth table specifies design output values as functions of other signals in a tabular form. The general syntax for a truth table varies depending on the type of outputs that are used in the design. For designs with combinatorial outputs, the general syntax is

```
truth_table ([inputs] -> [outputs])
        [input state 1] -> [output state 1];
        [input state 2] -> [output state 2];
            .
            .
            .
        [input state 3] -> [output state 3];
```

where outputs are the output signals of the design; inputs are the signals that make up the logic functions of the outputs; and the input and output states specify the output values for a given set of input values. For example, the truth table for a 3 input and gate with inputs A, B, and C and output OUT would look like:

```
truth_table ([A, B, C] -> [OUT])
        [0,  0,  0] -> [   0   ];
        [0,  0,  1] -> [   0   ];
        [0,  1,  0] -> [   0   ];
        [0,  1,  1] -> [   0   ];
        [1,  0,  0] -> [   0   ];
        [1,  0,  1] -> [   0   ];
        [1,  1,  0] -> [   0   ];
        [1,  1,  1] -> [   1   ];.
```

Designs with registered outputs have a general syntax of

```
truth_table ([inputs] :> [outputs])
        [input state 1] :> [output state 1];
        [input state 2] :> [output state 2];
            .
            .
            .
```

```
                [input state 3] :> [output state 3];
```

where the only difference from combinatorial designs is the :> instead of the -> in-between the inputs and outputs. For designs that have both combinatorial and registered outputs, the syntax is

```
truth_table ([inputs]        :> [reg_outputs]          -> [comb_outputs])
            [input state 1] :> [reg_output state 1] -> [comb_output state 1];
            [input state 2] :> [reg_output state 2] -> [comb_output state 2];
                .
                .
                .
            [input state 3] :> [reg_output state 3] -> [comb_output state 3] ;
```

where reg_outputs represent the registered design outputs, and comb_outputs are the combinatorial design outputs.

Truth tables must have parenthesis around the signal names, and there must be a semicolon after each line of the table. To shorten truth table entries, PHDL supports "don't care" conditions which may be entered in place of signal states. This eliminates the need to specify every possible signal state when the output values will be the same for several different input states. The "don't care" condition is entered as a .X. in place of the 1 or 0 that would have normally been entered. The example that follows illustrates how to use "don't cares".

The following example is a second implementation of the T-Bird Tail Lights design that uses a combination of a state diagram and a truth table. In this design, the state diagram controls only the state transition logic, and the truth table determines the output of the design by taking into account both the inputs and the current state. Note the use of "don't cares" in the truth table.

```
MODULE tbird

TITLE 'thunderbird tailight demo'

BRAKE pin ;
TL pin ;
TR pin ;
CLK pin ;

L1..L3 pin istype 'reg';
R1..R3 pin istype 'reg';
q2..q0 pin istype 'reg' ;

out = [L3, L2, L1, R1, R2, R3];

sreg = [q2, q1, q0] ;

s0 = [0, 0, 0] ;
s1 = [0, 0, 1] ;
s2 = [0, 1, 0] ;
s3 = [0, 1, 1] ;
s4 = [1, 0, 0] ;
s5 = [1, 0, 1] ;
s6 = [1, 1, 0] ;
s7 = [1, 1, 1] ;

equations

sreg.clk = CLK ;

out.clk = CLK ;

state_diagram sreg ;

state s0:
  if (TL ) then s1
  else if (TR) then s4
  else s0;

state s1:
  if (TL ) then s2
  else if (TR) then s4
  else s0;

state s2:
```

```
  if (TL ) then s3
  else if (TR) then s4
  else s0;

state s3:
  if (TL ) then s0
  else if (TR) then s4
  else s0;

state s4:
  if (TR) then s5
  else if (TL ) then s1
  else s0;

state s5:
  if (TR) then s6
  else if (TL ) then s1
  else s0;

state s6:
  if (TR) then s0
  else if (TL ) then s1
  else s0;

state s7:
  goto s0;        "Illegal state recovery

truth_table ([sreg.fb, BRAKE,  TR, TL] :> [L3, L2, L1, R1, R2, R3])
              [     s0   ,   0  , .x., .x.] :> [ 0,  0,  0,  0,  0,  0];
              [  s0   ,   1  ,   0,   0] :> [ 1,  1,  1,  1,  1,  1];
              [  s0   ,   1  ,   0,   1] :> [ 0,  0,  0,  1,  1,  1];
              [  s0   ,   1  ,   1,   0] :> [ 1,  1,  1,  0,  0,  0];
              [  s1   ,   0  , .x., .x.] :> [ 0,  0,  1,  0,  0,  0];
              [  s1   ,   1  , .x., .x.] :> [ 0,  0,  1,  1,  1,  1];
              [  s2   ,   0  , .x., .x.] :> [ 0,  1,  1,  0,  0,  0];
              [  s2   ,   1  , .x., .x.] :> [ 0,  1,  1,  1,  1,  1];
              [  s3   ,   0  , .x., .x.] :> [ 1,  1,  1,  0,  0,  0];
              [  s3   ,   1  , .x., .x.] :> [ 1,  1,  1,  1,  1,  1];
              [  s4   ,   0  , .x., .x.] :> [ 0,  0,  0,  1,  0,  0];
              [  s4   ,   1  , .x., .x.] :> [ 1,  1,  1,  1,  0,  0];
              [  s5   ,   0  , .x., .x.] :> [ 0,  0,  0,  1,  1,  0];
              [  s5   ,   1  , .x., .x.] :> [ 1,  1,  1,  1,  1,  0];
              [  s6   ,   0  , .x., .x.] :> [ 0,  0,  0,  1,  1,  1];
              [  s6   ,   1  , .x., .x.] :> [ 1,  1,  1,  1,  1,  1];
END
```

Chapter 6 *Simulating XPLA™ Designs*

The **XPLA™ Designer** simulator is a graphical, interactive design analysis tool. It allows the user to input test vectors directly from the waveform screen, move signal edges with the touch of a mouse button, measure timing delays, and many other features normally found on expensive high-end systems. The graphical control of stimulus signals allows designs to be thoroughly simulated in much less time than conventional simulators that require textual test vector entry.

Organization

This section is comprised of four (4) parts:

- **Simulation process**
- **Description of Simulator menus**
- **Practice design simulation**
- **SCL (Simulation Control Language) brief**

Simulation process

The XPLA™ Designer simulation engine performs functional and timing simulation of the input design using signal stimuli defined in the Simulation Control Language, (SCL) file. This file is generated when the user defines signal characteristics from the waveform window or when the information is input in text form. Inputs to the simulator are the SCL file, which also contains timing information, and the user design file in binary form. Results from the simulation can be directly displayed in the waveform viewer and output to any Windows™ supported printer.

The SCL file and the binary design file are combined to produce an event list. Figure 30 illustrates this process.

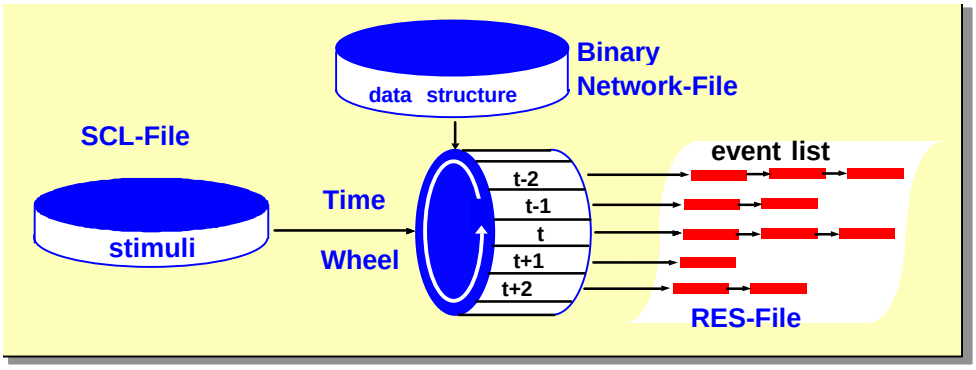


Figure 30

The simulator utilizes five logic levels when representing waveforms.

- **Low**
- **High**
- **Unknown**
- **Tri-State**
- **Undefined**

The SCL and Waveform viewer representation of these values are described in Figure 31 and Figure 32. The waveform viewer displays four of the five values, excluding the undefined state.

State	Symbol	Meaning
Low	0	Logic state is low, (false)
High	1	Logic state is high, (true)
Unknown	*	Logic state is not known, but either low or high.
Tri-state	3	High impedance or floating state
Undefined	#	Logic state is between low and high, indicating an unstable level

Figure 31

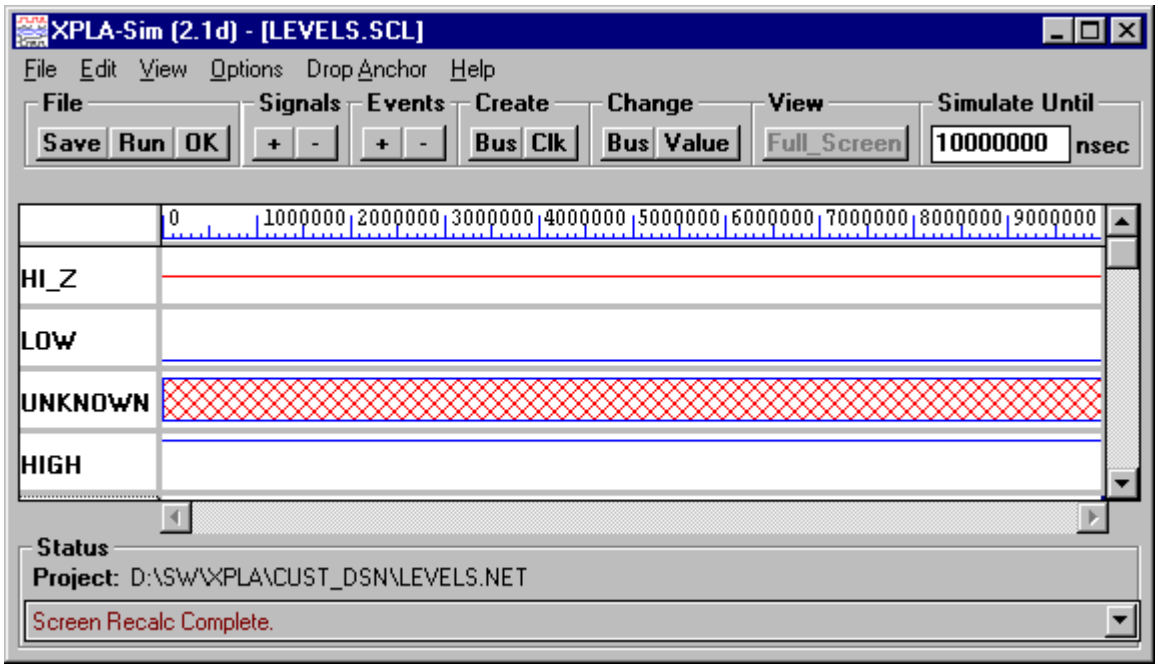


Figure 32

Combining the user control of input stimuli with the data files produces waveform simulations of considerable complexity in a very short time. Figure 33 represents an example of the waveform viewer output.

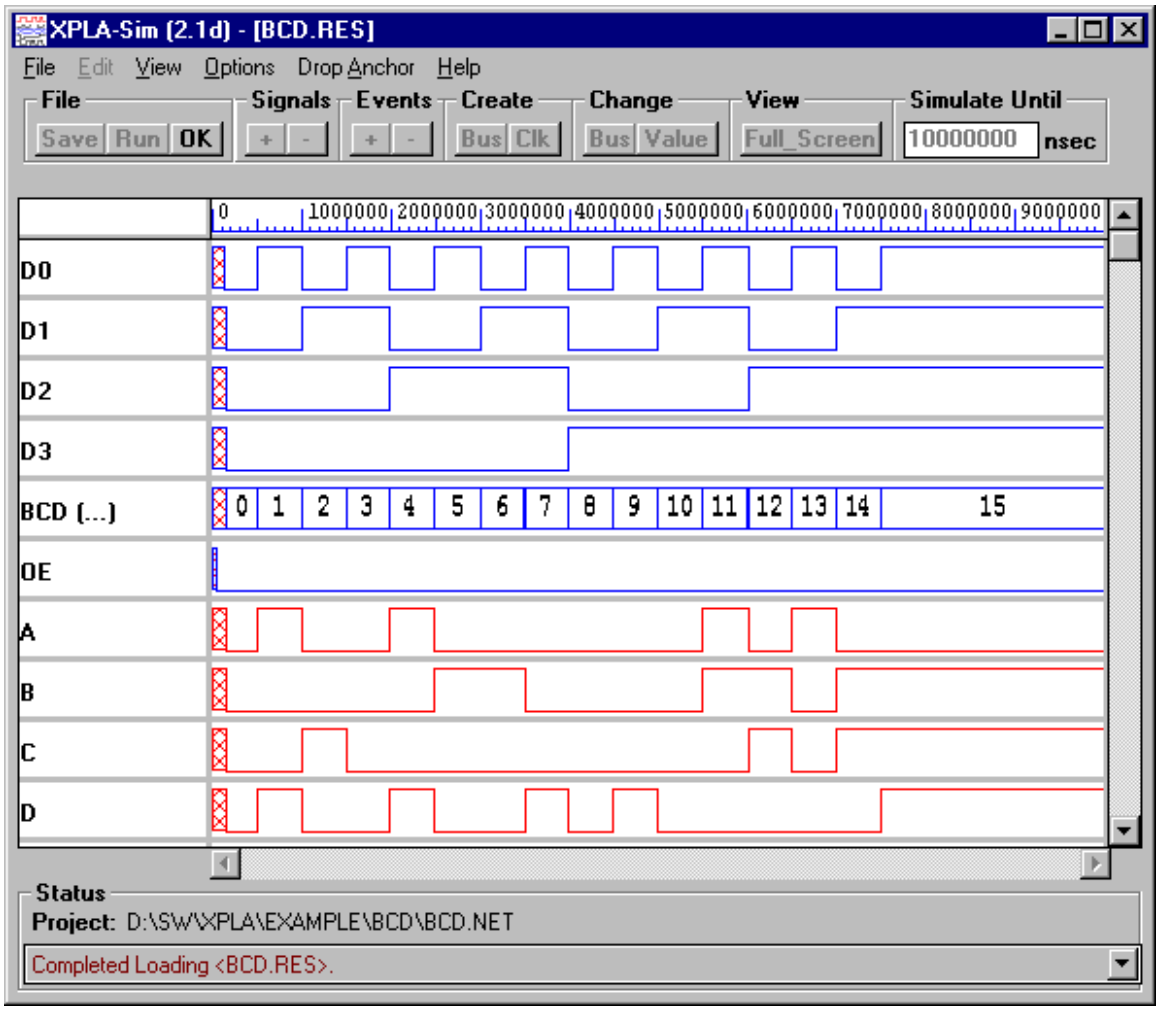


Figure 33

Description of Simulator menus

As can be seen in Figure 34, the **XPLA™ Designer** simulator operation is controlled via the waveform viewer menu and toolbar.

Figure 34

The menu bar consist of the following items:

- **File**
- **Edit**
- **View**
- **Options**
- **Drop Anchor**
- **Help**

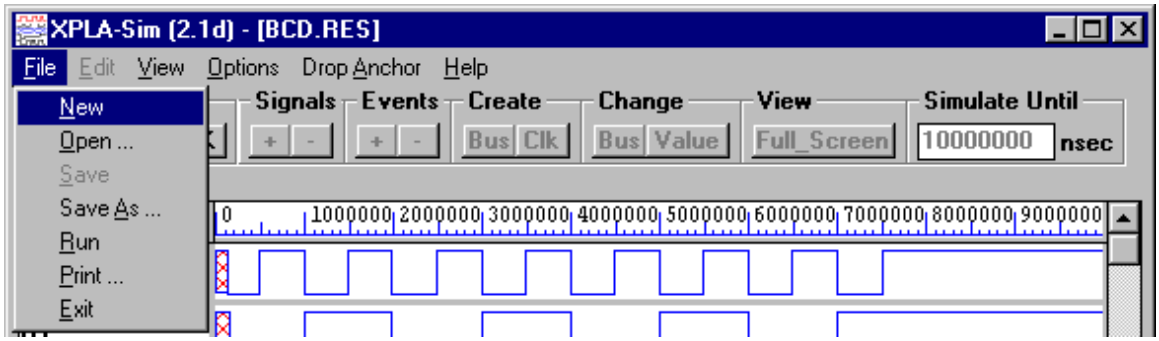


Figure 35

Under **File** there are eight additional items which are:

New

This command allows the operator to start a simulator waveform viewer window without using the SCL file generated when a particular design is compiled.

Open

This command allows the operator to simulate an existing compiled design instead of the one listed under 'Filename' in the Design Panel of XPLA™ Designer. You may also explicitly open a functional simulation by opening the .mod file created for your design, or an AC timing simulation by opening the .net file.

Save

This command allows the operator to save the results of the currently defined stimuli displayed in the waveform viewer to the SCL file.

Save As

This command allows the operator to save the results of the currently defined stimuli displayed in the waveform viewer under a different SCL file name. One way this command can be utilized is the saving of successive design iteration simulations or to keep multiple sets of stimuli.

Run

This command is used to start the simulation process upon the design listed in brackets at the top of the waveform viewer window. Figure 36 shows the design Count2.scl in brackets. A design that has been loaded but not run will carry the suffix .scl. After the simulator has run the suffix will be .res.

Print

This command will print the contents of the waveform viewer screen to the printer set as default under the windows printer menu.

Page Setup

This command allows the operator to control the orientation of the printed waveform viewer window.

Exit

This command will terminate the waveform viewer window and take the operator back to the Design Panel window of XPLA™ Designer.[3]

[3] *THIS IS A 100% MATTER PRODUCT: In the Unlikely Event That This Merchandise Should Contact Antimatter in Any Form, a Catastrophic Explosion Will Result.*

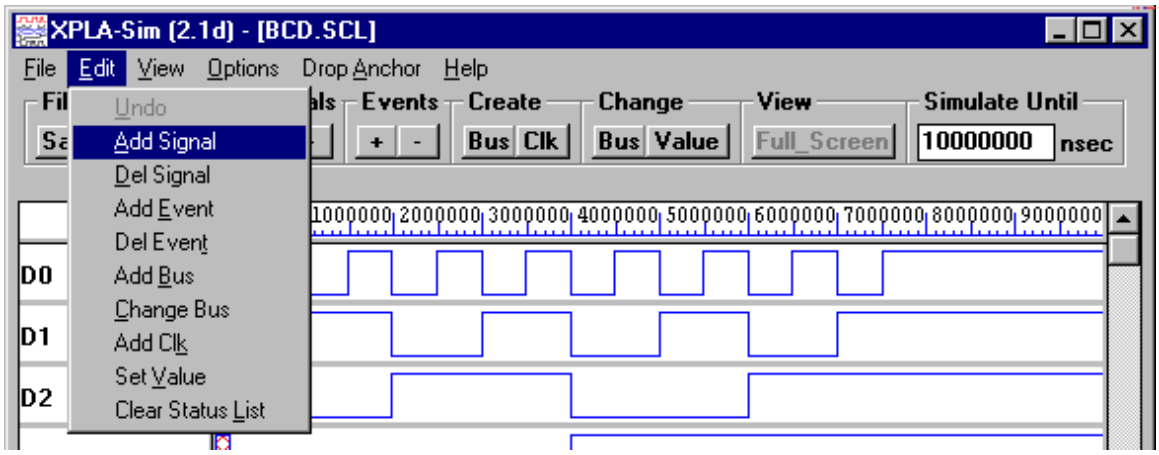


Figure 36

Under **Edit** there are ten additional items which are:

Undo

This command will allow the operator to delete the last entered command.

Add Signal

This command will allow the operator to add a signal to the waveform viewer window. An **Add Signal** window will open and a list of signals available to add will appear. In the waveform viewer window, click in the row of signal names at the location where you want the new signal to appear. You will be prompted for a new signal name. Type the name of the signal, or select a name from the list provided. If the name you type is not the name of an available signal it will be assumed to be an internal node and automatically set to be an output. When you are finished click the **Done** button and the new signal name will appear above location you selected. All signal names below the point you selected will be pushed down the waveform viewer window. A signal can have any valid name of eight characters or less. See Figure 37 for details.

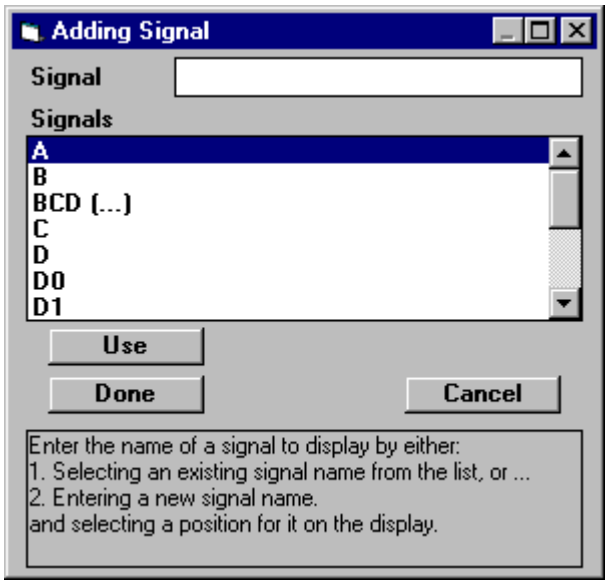


Figure 37

Del Signal

This command will allow the operator to delete a signal from the waveform window. When this command is selected, the message, "Click on the row which you wish to delete" will appear in the project status window of the waveform viewer. The signal will be deleted from view if you answer yes to the safety prompt, otherwise it will remain visible. The project status box is located at the bottom of the waveform viewer window, (see Figure 38).

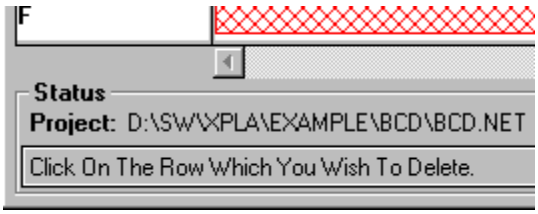


Figure 38

Add Event

This command will allow the operator to add signal events, i.e. logical highs, or logical lows, to a named signal. Click on the signal at the time position at which you want the

new transition to occur. If the signal was high, it will go low. If the signal was low, it will go high. If the value of the signal is unknown or tri-state, it will go low. Continue adding events to the original signal, or different signals, until you are finished, then click the **OK** button. When selected, the project status window will display instructions on how to add events to signal lines and a large upward pointing arrow will appear indicating that you are in the **ADD EVENT** mode. Figure 39 is an example of what you should see upon entering this mode.

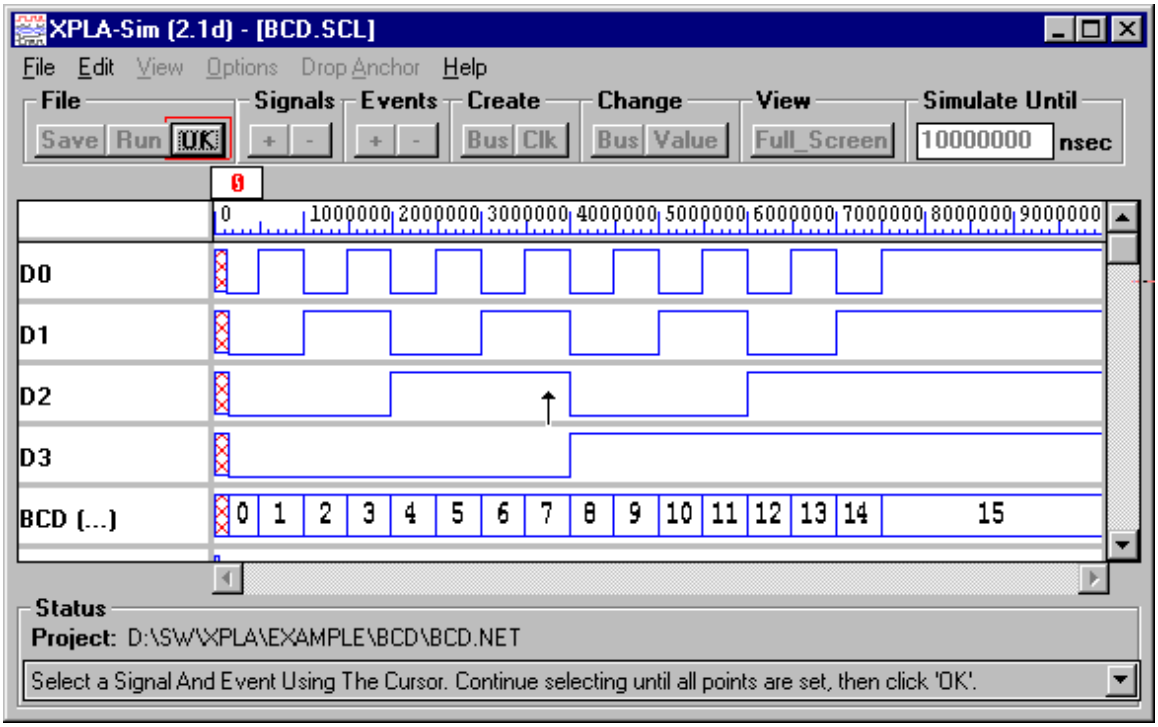


Figure 39

Del Event

This command will allow the operator to delete signal events, i.e. logical highs, or logical lows, to a named signal. Click to the left of the transition you want to delete on the row of the selected signal. The transition will be deleted. When selected, the project status window will display instructions on how to delete events from signal lines. Figure 40 shows project status information for the **Del Event** option.

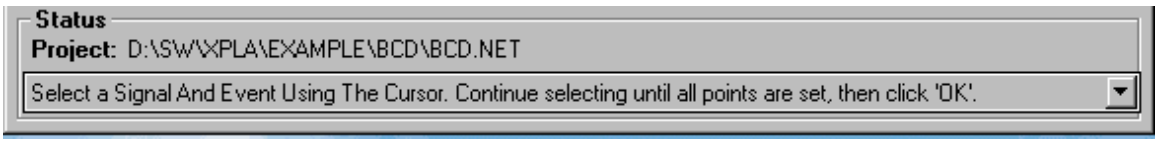

Status
Project: D:\SW\XPLA\EXAMPLE\BCD\BCD.NET
Select a Signal And Event Using The Cursor. Continue selecting until all points are set, then click 'OK'.

Figure 40

Add Bus

This command will allow the operator to create a bus signal from available signals and display it in the waveform viewer window. Click at the position in the row of signals displayed in the waveform viewer where you want the bus signal to appear. In the **adding bus** window, a list of available signals appear on the left side of the window. Highlighting a signal and clicking on the **add button**, or double clicking on the signal name, will transfer that signal to the **bus elements** box on the right side of the window. Select the bus elements you wish to include up to a maximum of 31. If a signal is inadvertently added to the bus elements box, it can be deleted by clicking on the **delete** button inside the window. You can also cancel the entire bus operation by clicking on the **cancel button.** When all the desired signals are added to the bus element box and the bus has a name; click the **done** button inside the **adding bus** window. The new bus signal will immediately be displayed above the location you selected in the signal name column. A bus will be automatically made an output if any of the elements are an output or an internal node. A bus element may not be another bus. See Figure 41 for details.

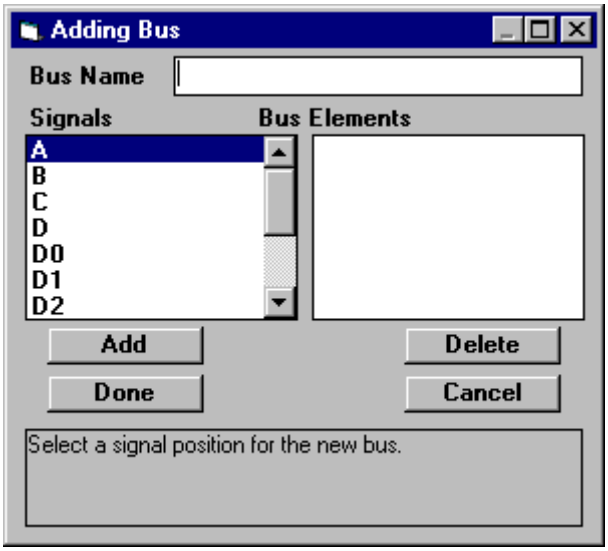


Figure 41

Change Bus

This command allows the operator to edit the bus signal elements. When you click the **change bus** option, an **editing bus** window will appear in the waveform viewer window. Click on the bus signal in the waveform viewer window you wish to edit. You may add signals to the bus or delete them from this window. Note that this command does not allow you to edit the value on a bus - only those elements that constitute a bus. The individual elements of a bus must be edited to change the bus value. See the section on

adding a bus, (page 73), for proper bus signal procedures. Figure 42 depicts the **editing bus** window.

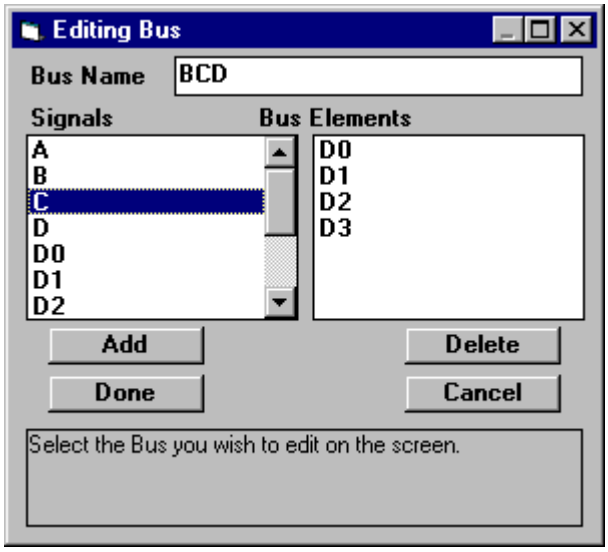


Figure 42

Add Clk

This command will allow the operator to add a clock to a signal name. A **clock settings** window will appear in the waveform viewer window. Select the start value, cycle length, and duty cycle you want, then click the **accept** button in the clock settings window. Select a signal name from the waveform viewer window by clicking on it. The clock setting you selected will appear to the right of the selected signal name. Figure 43 illustrates the procedure.

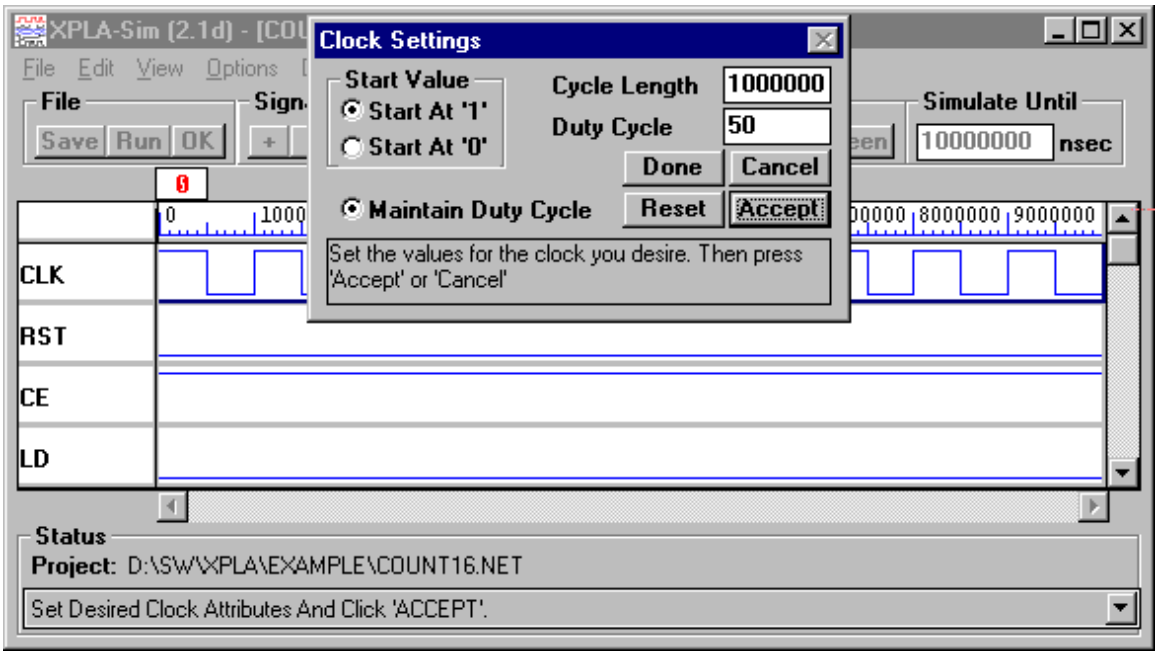


Figure 43

Set Value

This command allows the operator to change the value of a signal at the location in the signal waveform selected by the cursor position. A **values** window will appear in the waveform viewer window when you select the location on the waveform you wish to edit. Select the value you wish the signal to be set to by clicking on the value displayed in the **values** window. Continue editing signals until you are satisfied, then click **OK** in the waveform viewer window.

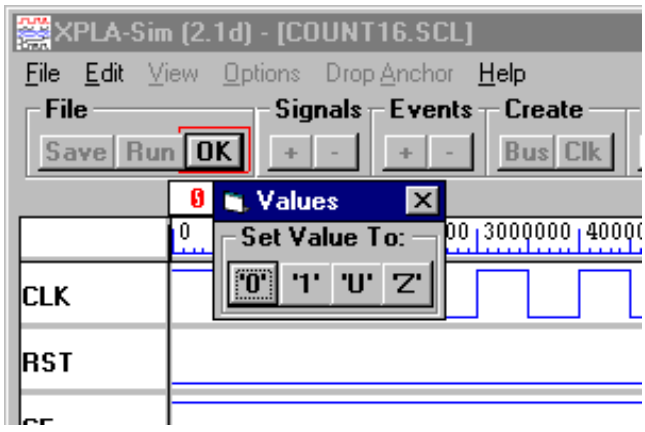


Figure 44

Clear Status List

This command allows the operator to clear the **status project:** window located at the bottom of the waveform viewer window. See Figure 40 for an example of the status project: box.

Under **View** there are three additional items.

Zoom Out

Use this command to incrementally compress the waveform viewer time scale.

Zoom In

Use this command to incrementally expand the waveform viewer time scale.

Full Screen

Use this command to view the entire waveform viewer time scale.

NOTE: There are three ways to Zoom in or out. You can:

1. Position the cursor at the spot in the waveform viewer window where you want the center of the display to be. Double click with the left mouse button to zoom in. The display will center on the position you selected if possible. You can double click the right mouse button to zoom out.

2. Determine the minimum and maximum time you want to display. Position the cursor in the time ruler portion of the waveform viewer window. Click and hold the left mouse button at the point where you want the new time to start. Drag the cursor to the point where you want the time to stop. The area between the start and end points will be highlighted. When you release the mouse button, the time you selected will be displayed. See Figure 45 for an example of this process.

3. You can toggle between a zoomed and non-zoomed condition by clicking on the **View** button in the toolbar. If you are currently zoomed in the button will read **Full Screen**. Clicking on the button will take you to a non-zoomed condition. When you click the button it will change to **Zoom Back**. Clicking the button will take you back to the previous zoom condition. Figure 46 and Figure 47 illustrate this function.

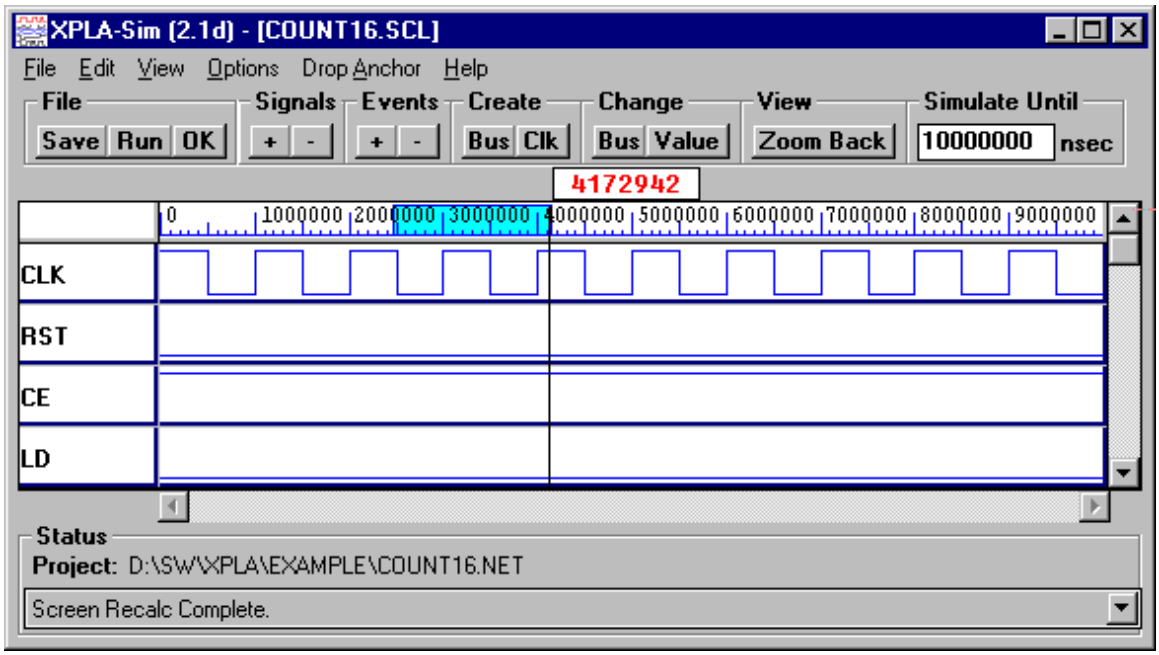


Figure 45

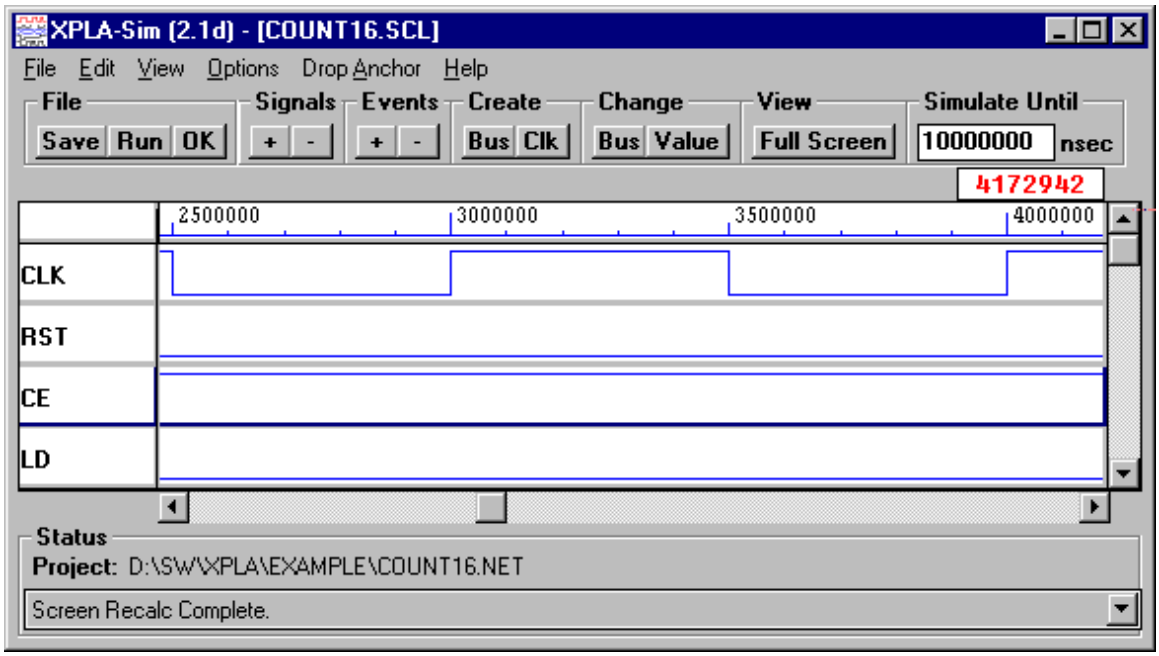


Figure 46

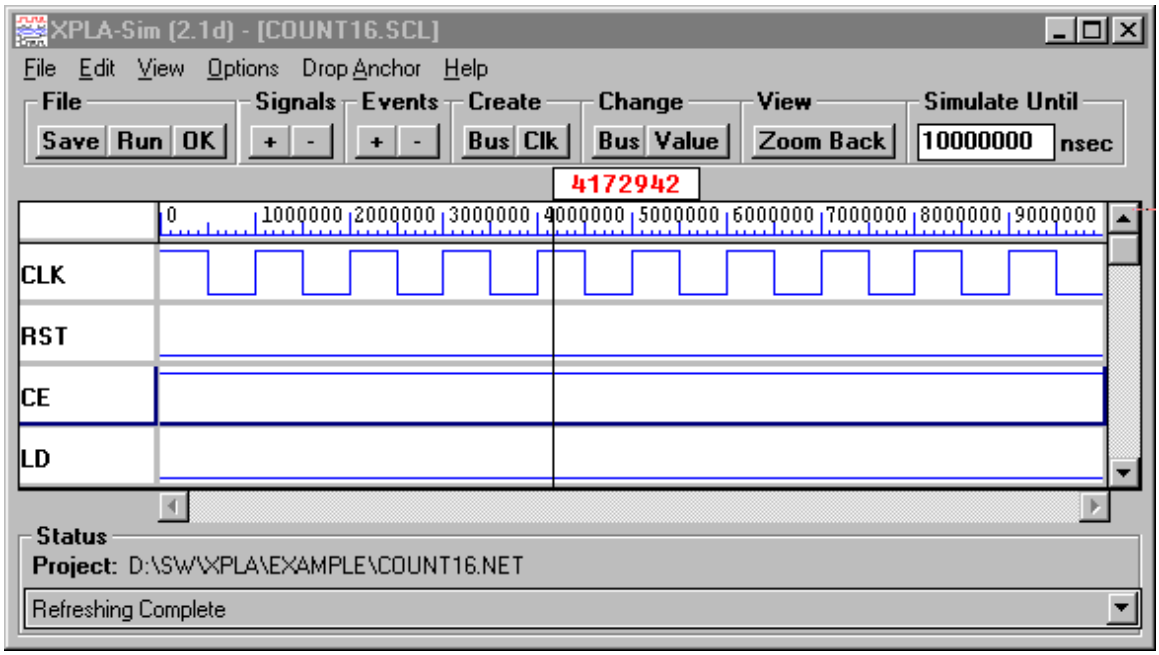


Figure 47

Under **Options** there are the three additional items which are:

Set Signal Height

This command allows the operator to scale the signal display height. Using this command you can adjust the number of signals displayed in the window. When selecting this item a **signal height** window will appear in the waveform viewer window (see Figure 48). Set the height to a value between 10 and 50 inclusive. The value you select is saved in the project .ini file when the program is exited and will be the start up value when the program is invoked. Figure 49 and Figure 50 are examples how this command works.

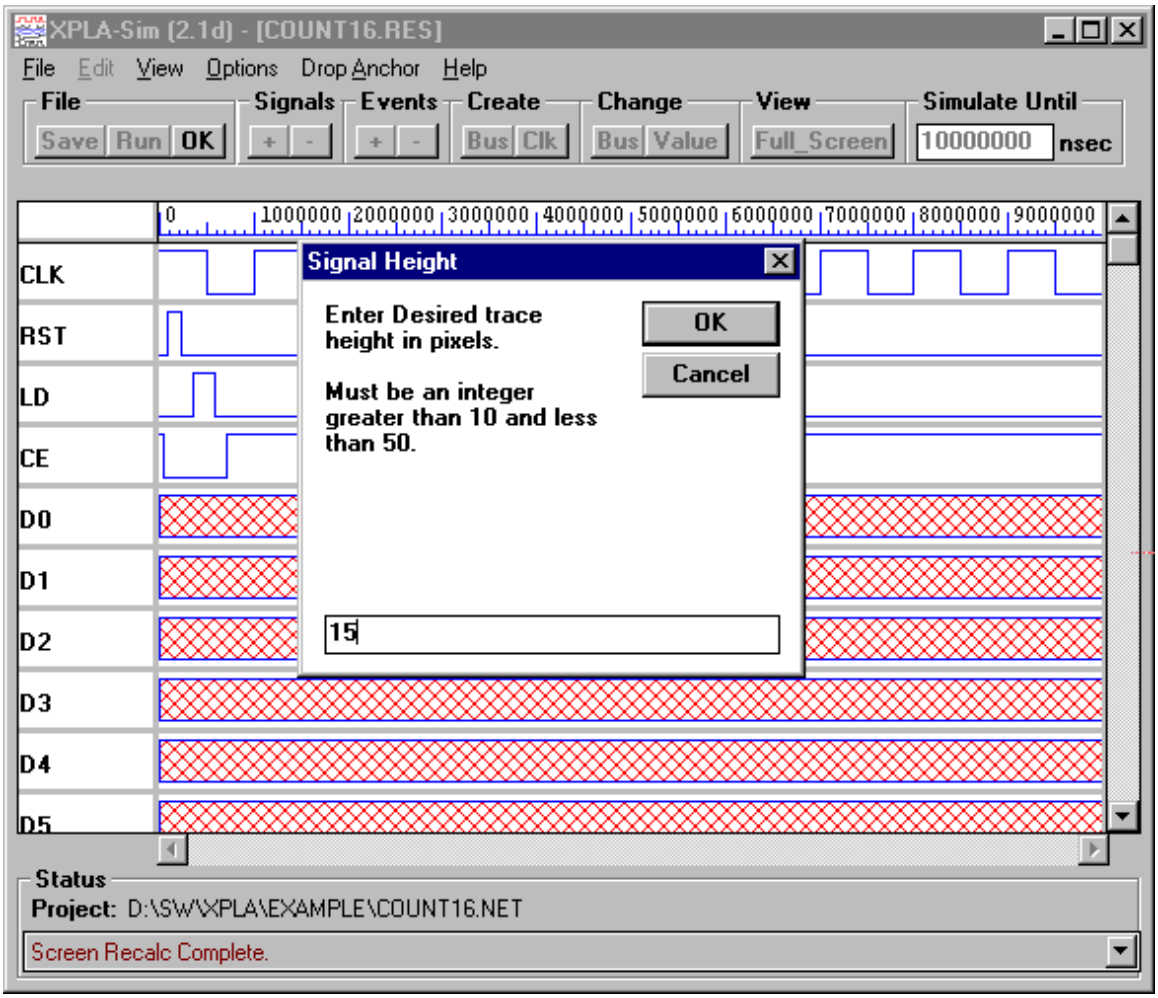

XPLA-Sim (2.1d) - [COUNT16.RES]
File Edit View Options Drop Anchor Help
File
Save Run OK
Signals + -
Events + -
Create Bus Clk
Change Bus Value
View Full_Screen
Simulate Until
10000000 nsec
0 1000000 2000000 3000000 4000000 5000000 6000000 7000000 8000000 9000000
CLK
RST
LD
CE
D0
D1
D2
D3
D4
D5
Signal Height
Enter Desired trace height in pixels.
Must be an integer greater than 10 and less than 50.
OK
Cancel
15
Status
Project: D:\SW\XPLA\EXAMPLE\COUNT16.NET
Screen Recalc Complete.

Figure 48

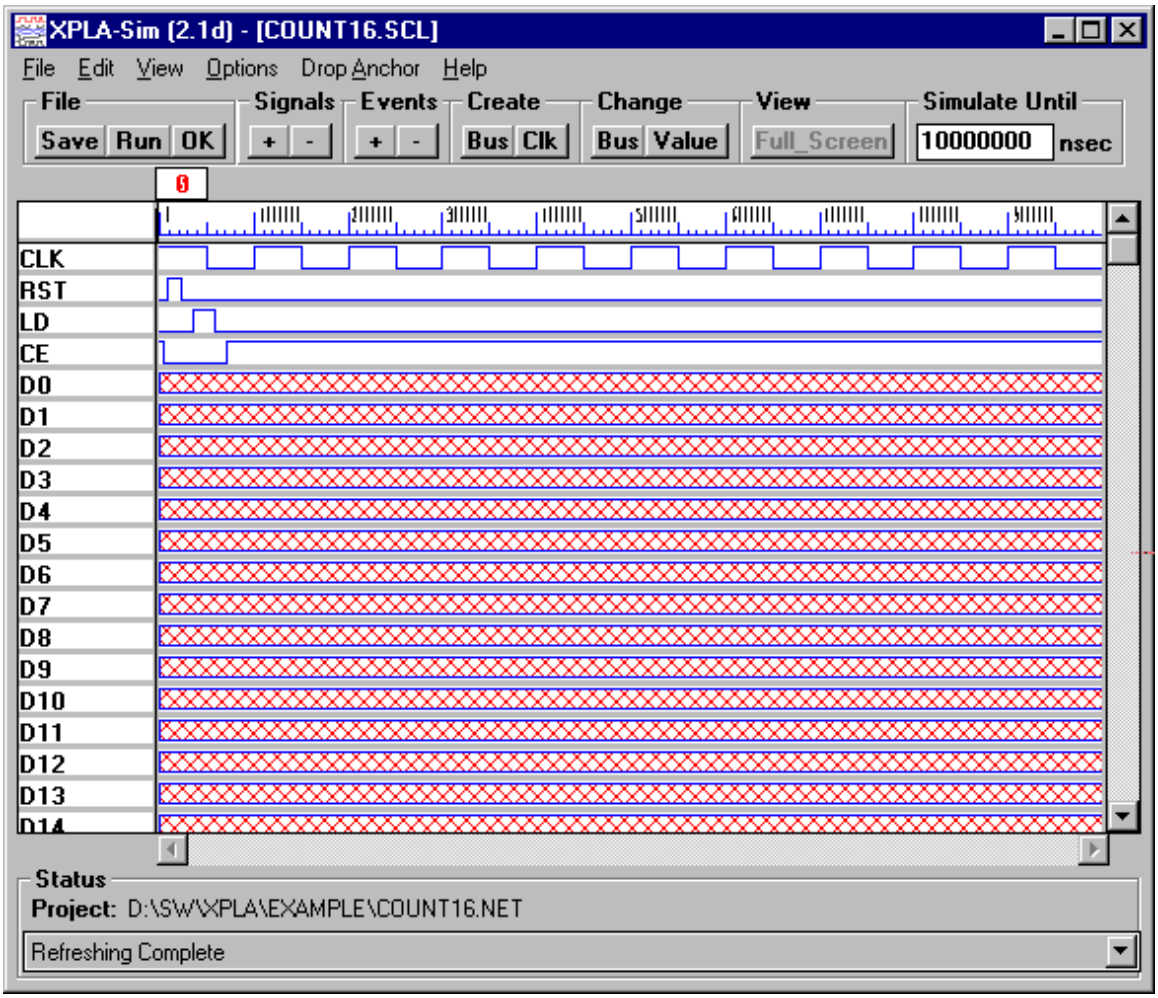

XPLA-Sim (2.1d) - [COUNT16.SCL]
File Edit View Options Drop Anchor Help
File
Save Run OK
Signals
+ -
Events
+ -
Create
Bus Clk
Change
Bus Value
View
Full_Screen
Simulate Until
10000000 nsec
CLK
RST
LD
CE
D0
D1
D2
D3
D4
D5
D6
D7
D8
D9
D10
D11
D12
D13
D14
Status
Project: D:\SW\XPLA\EXAMPLE\COUNT16.NET
Refreshing Complete

Figure 49

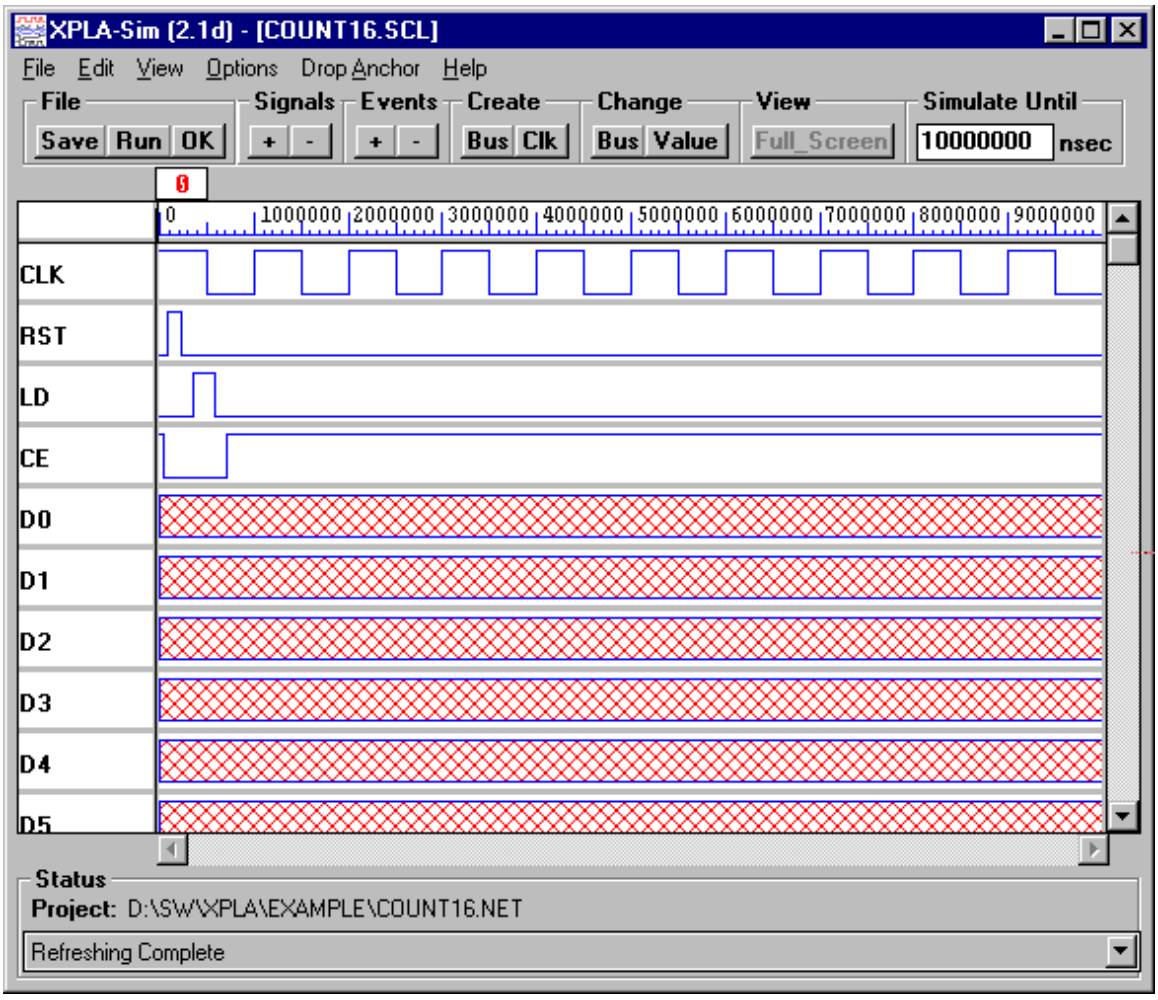


Figure 50

Transition Check

This command allows the operator to check for signal transitions during the simulation run. The results of the check can be viewed in the **project status** display window. Figure 51 one is an example of this option's output in the project status window.

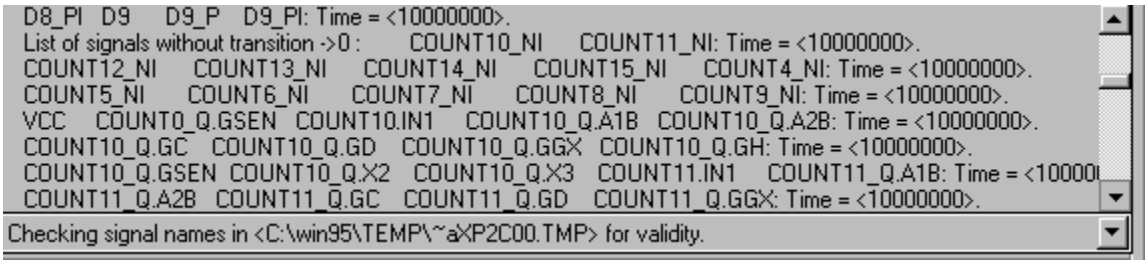


Figure 51

Auto Save SCL on Run

This command allows the operator to automatically save the most recent SCL file when the simulation is run. The default for this item is enabled. If it is disabled, the current waveforms will not be saved and the simulation will be run using the most recently loaded SCL file.

Drop Anchor

The **Drop Anchor** menu item, when invoked, activates the waveform viewer time measurement function. See Figure 52 for an example of this invocation. Select the starting point of the measurement and position your cursor just to the left of that point. Click and hold down the left mouse button. After a short delay, several things will happen:
A red circular anchor mark will appear on the event you have selected

1. A red vertical red line will appear, bisecting the circle, and spanning the entire waveform viewer window
2. The **Drop Anchor** menu item will change to **Raise Anchor**
3. The color of the selected signal name will change from black to red
4. A box will appear above the time ruler that contains two numbers. The number in blue indicates the time in nanoseconds that separates the anchor point from the new cursor position, (note in Figure 53 that when you first invoke the anchor function, this number is zero). The number in red is the actual position on the time ruler where the cursor is located.

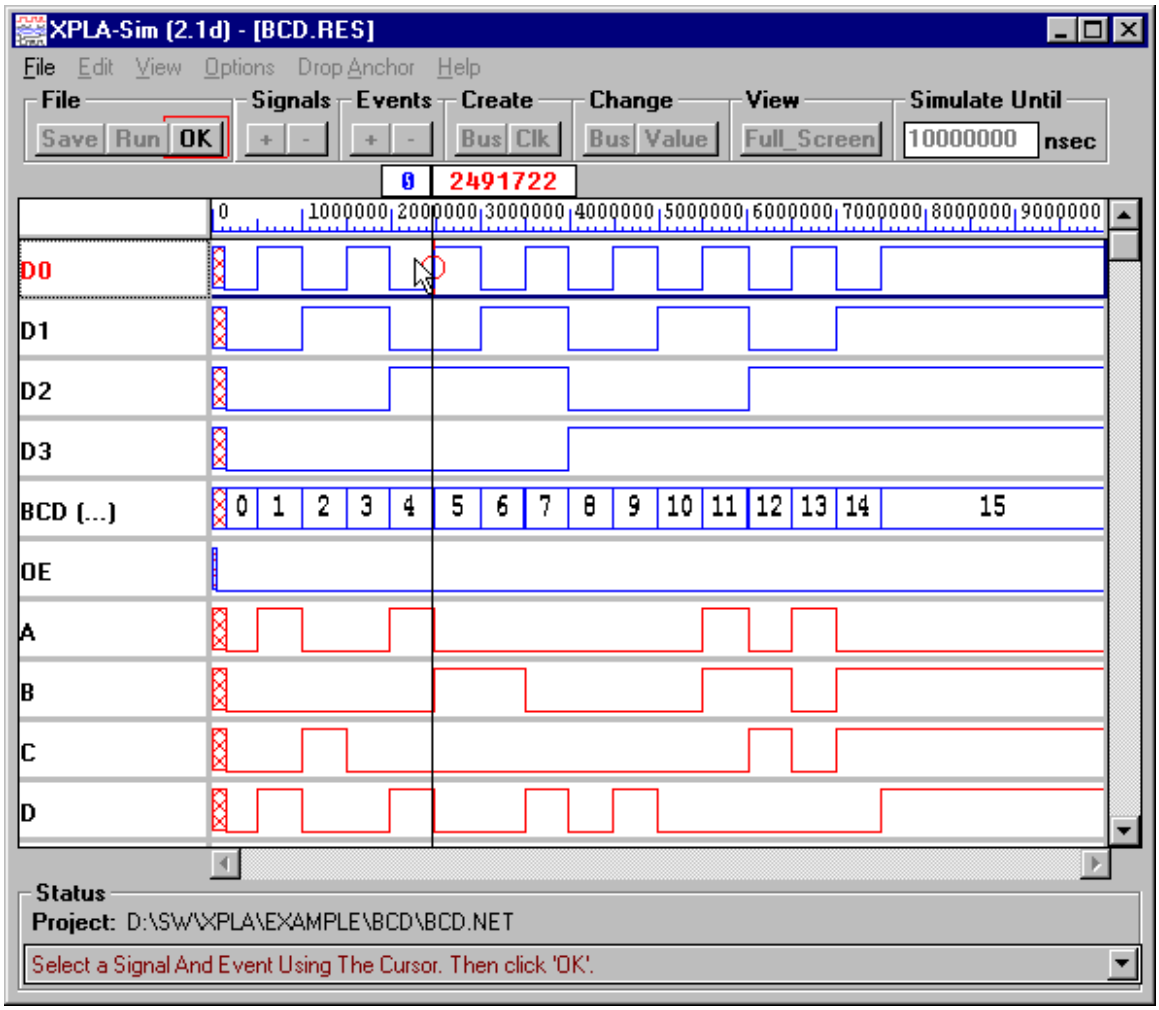


Figure 52

To measure the difference between the anchor point and anywhere to the right of that point; position the cursor to a new location in the waveform viewer and click the left mouse button. It may be necessary to expand the scale in order to measure signal transitions occurring close to one another. When you click on a location to measure, Several things will happen:

1. A black vertical line will appear, at the location you selected, spanning the entire waveform viewer window
2. The blue numbers will now indicate, in nanoseconds, the difference between the anchor point and the new cursor position.
3. The red numbers will indicate the new location on the time ruler where the cursor lies.
4. The red and blue numbers box will follow the location black vertical line as you measure different transitions within the waveform viewer window.

Figure 53 illustrates this delay measurement.

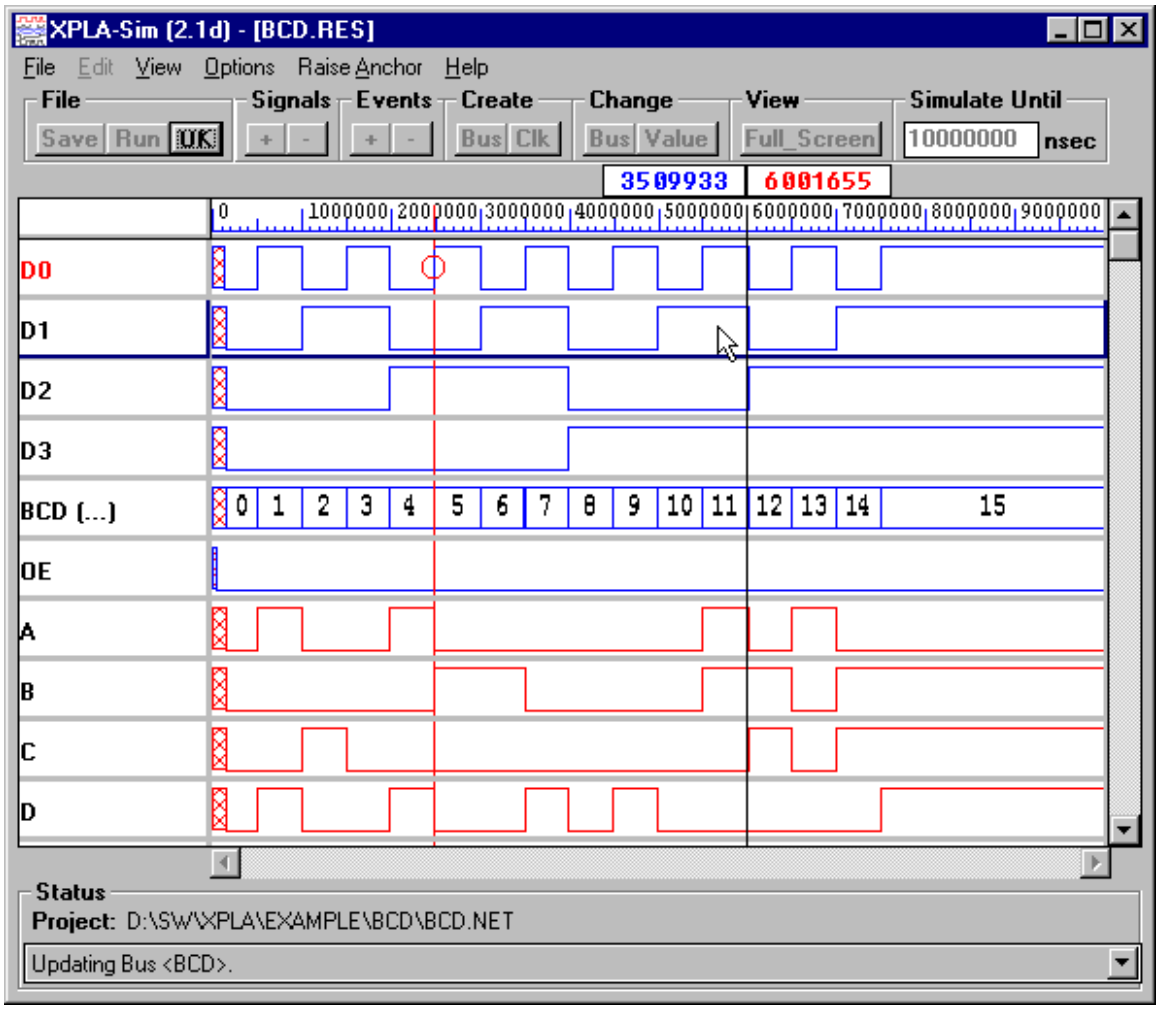


Figure 53

Raise Anchor

To delete the anchor, click on the **Raise Anchor** item in the menu bar. To select a different anchor point simply repeat the steps listed above.

The tool bar consist of the following items:

- File [**Save, Run, and OK]**
- Signals [**+, -]**

- Events **[+, -]**
- Create **[Bus, Clk]**
- Change **[Bus, Value**
- View **[Full Screen, Zoom Back]**
- Simulate **[xxxxxxxx nsec]**

Figure 54 depicts this arrangement. Figure 55 through Figure 68 show each function in greater detail.

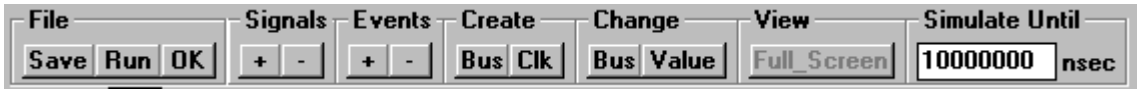


Figure 54

Press the File **Save** button to save the current **SCL** file.

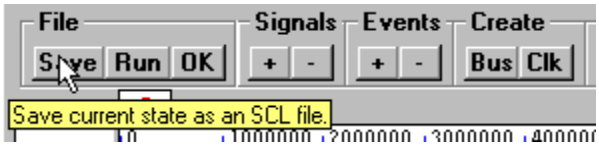


Figure 55

Press the File **Run** button to save the current waveform stimuli to the SCL file (assuming you have the autosave option on) and run the simulator.

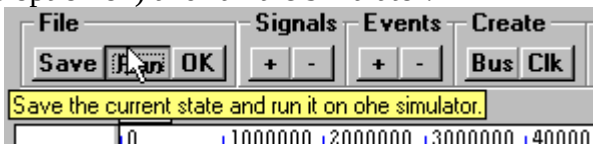


Figure 56

Press the File **OK** button to:

1. Redraw the waveform viewer window
2. End the **add/delete events** process
3. End the **value change** process

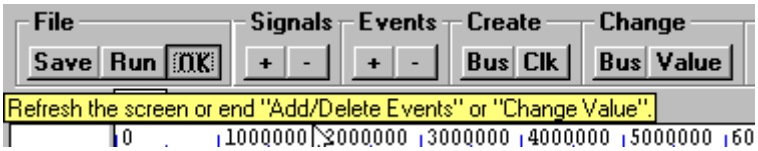


Figure 57

Press the Signals + button to add a signal to the waveform viewer window. See page 70 under **Add Signal** for a complete description of this function.

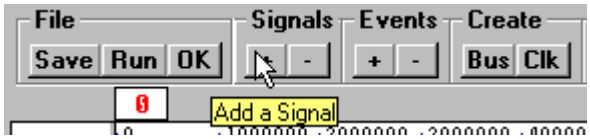


Figure 58

Press the Signals - button to remove a signal to the waveform viewer window. See page 71 under **Del Signal** for a complete description of this function.

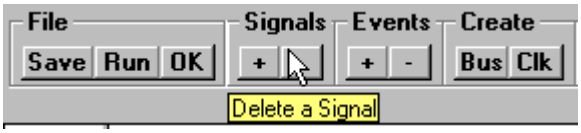


Figure 59

Press the Events + button to add a transition to a signal or signals displayed in the waveform viewer window. See page 71 under **Add Event** for a complete description of this function.

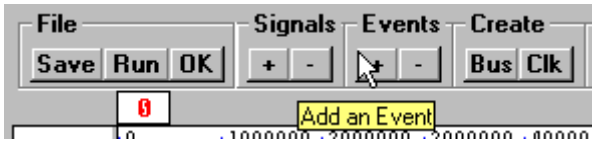


Figure 60

Press the Events - button to delete a transition from a signal or signals in the waveform viewer window. See page 72 under **Del Event** for a complete description of this function.

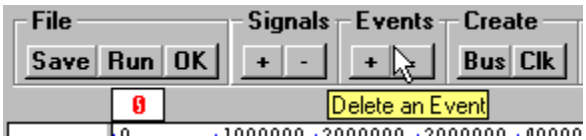


Figure 61

Press the Create **Bus** button to add a bus signal to the waveform viewer window. See page 73 under **Add Bus** for a complete description of this function.

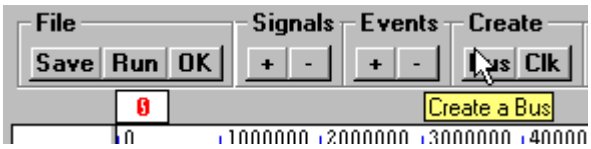


Figure 62

Press the Create **Clk** button to add a clock signal to the waveform viewer window. See page 74 under **Add Clk** for a complete description of this function.

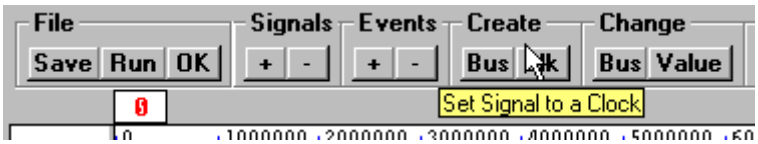


Figure 63

Press the Change **Bus** button to edit an existing bus in the waveform viewer window. See page 73 under **Change Bus** for a complete description of this function.

Figure 64

Press the Change **Value** button to edit the value of a signal in the waveform viewer window. See page 75 under **Set Value** for a complete description of this function.

Figure 65

Press the View **Full Screen/Zoom Back** button to switch between an normal or expanded view of the waveform viewer window. See page 76 under **View** for a complete description of this function.

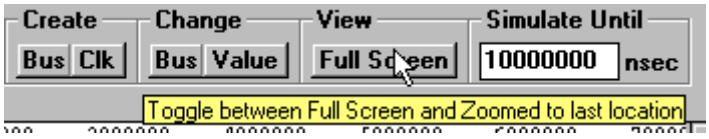


Figure 66

- **Simulation Length**

The **Simulate Until** window indicates the point at which the simulation ends. You can scale the waveform viewer window by adjusting this number. See Figure 67 and Figure 68 for the effect of this action on the identical waveform viewer window.

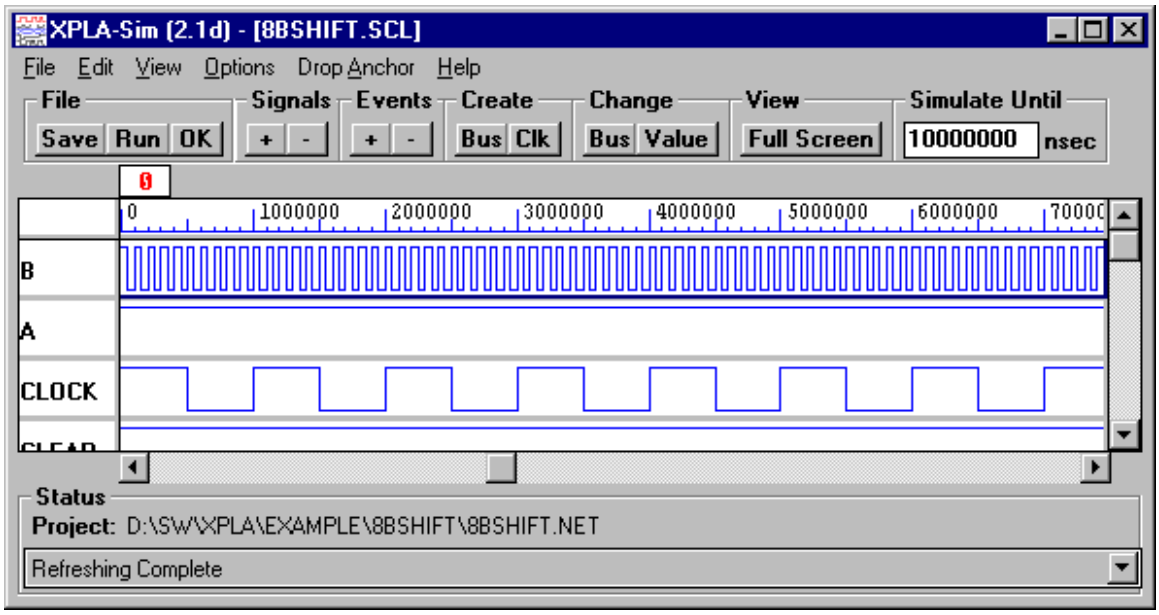


Figure 67

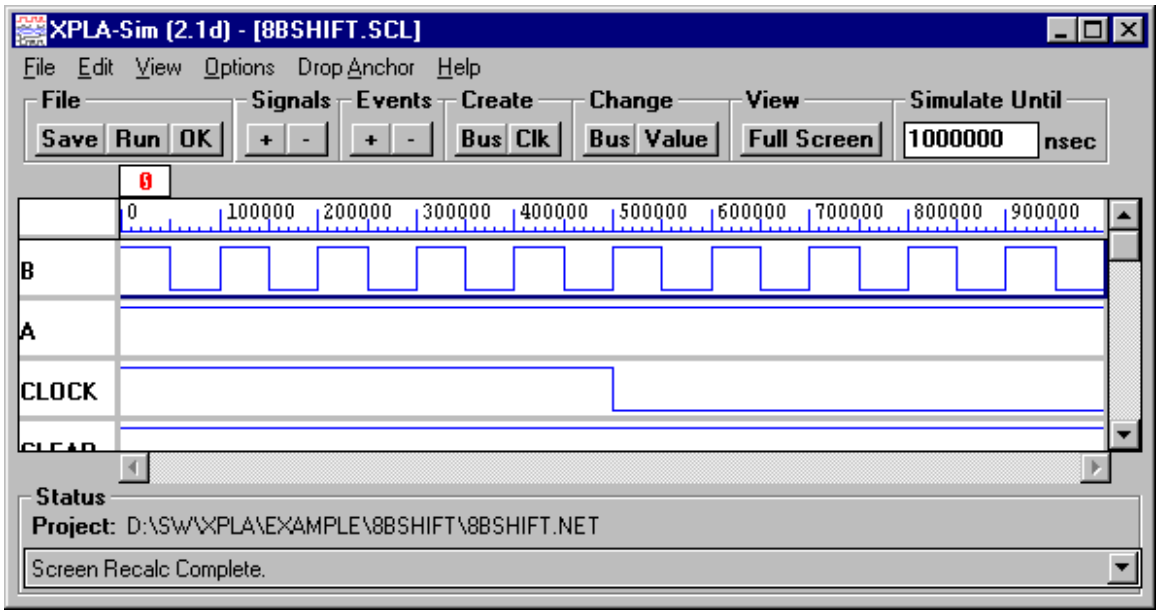


Figure 68

- **Practice Design Simulation**

In this section we will simulate the design that ships with XPLA™ Designer. The name of the design is **DEMO.PHD**. From the entry screen, the one that comes up when you start XPLA™ Designer, click on the word **DESIGN** in the menu bar located at the top of the window.

A new window will pop up with the following selections:

1. **New Design**
2. **Open Design**
3. **Exit**

Figure 69

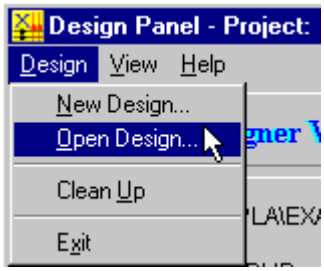


Figure 70

Click on the **Open Design** selection of the new window. (The **Clean-Up** option can be used later to delete intermediate files created by the compiler) An **Open Design** window will appear. Use this window to select **Demo.phd**, then click on the **OK** button.

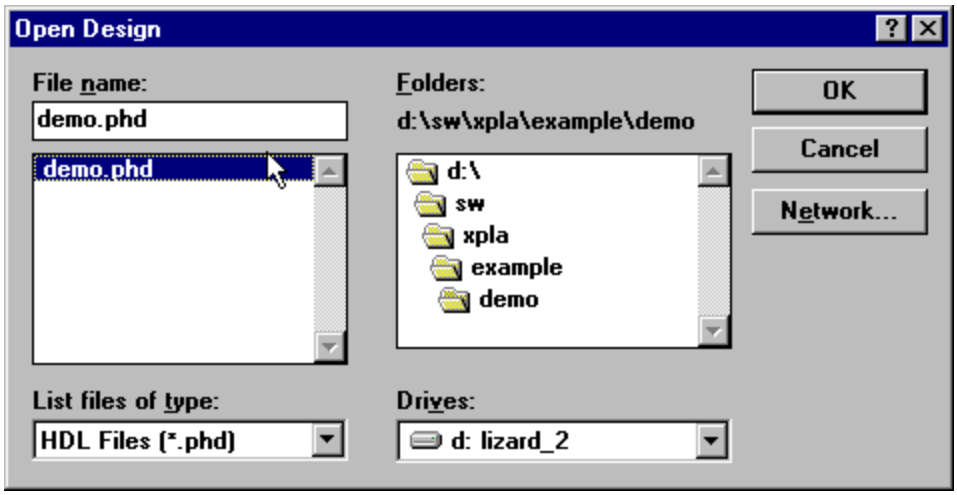


Figure 71

You will now be back to the main **XPLA™ Designer** window. The **Design Panel - Project:** line at the top of the window will now read **Demo**. Click on the **Compile..** button located at the bottom left side of the window. The design will be compiled and the result will be in a format the simulator can use.

Figure 72

Figure 73 shows the completed compilation process as indicated in the status window.

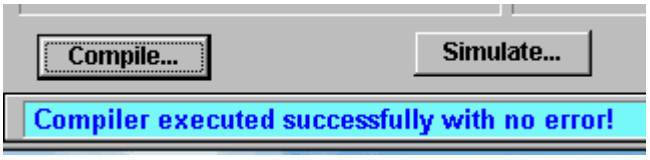


Figure 73

Now click on the **Simulate** button located at the bottom center of the **Design Panel** window. You should see a window very similar to the one depicted in Figure 74.

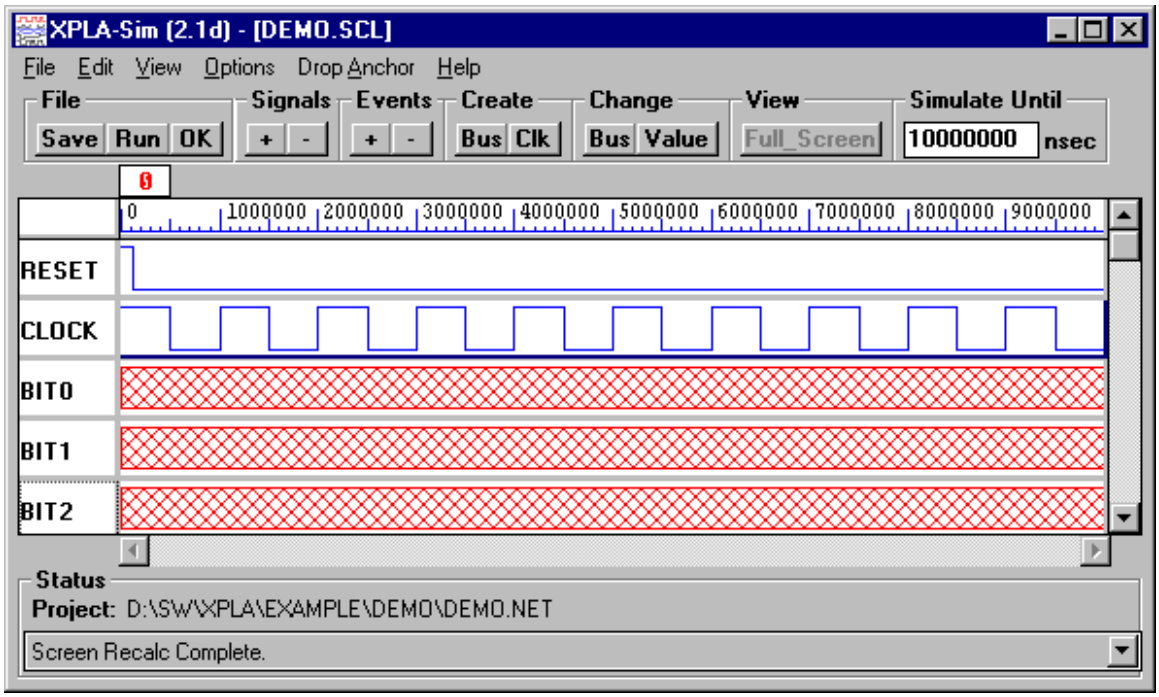


Figure 74

The Demo design comes with some stimuli already defined - let's add another reset pulse. Click on the **Events +** button Position the up arrow cursor about 3 clocks into the simulation on the **RESET** signal. Click the left mouse button. Position the cursor a millimeter to the right of the transition you have just created, and click the left mouse button again. The waveform should look like the following in Figure 75.

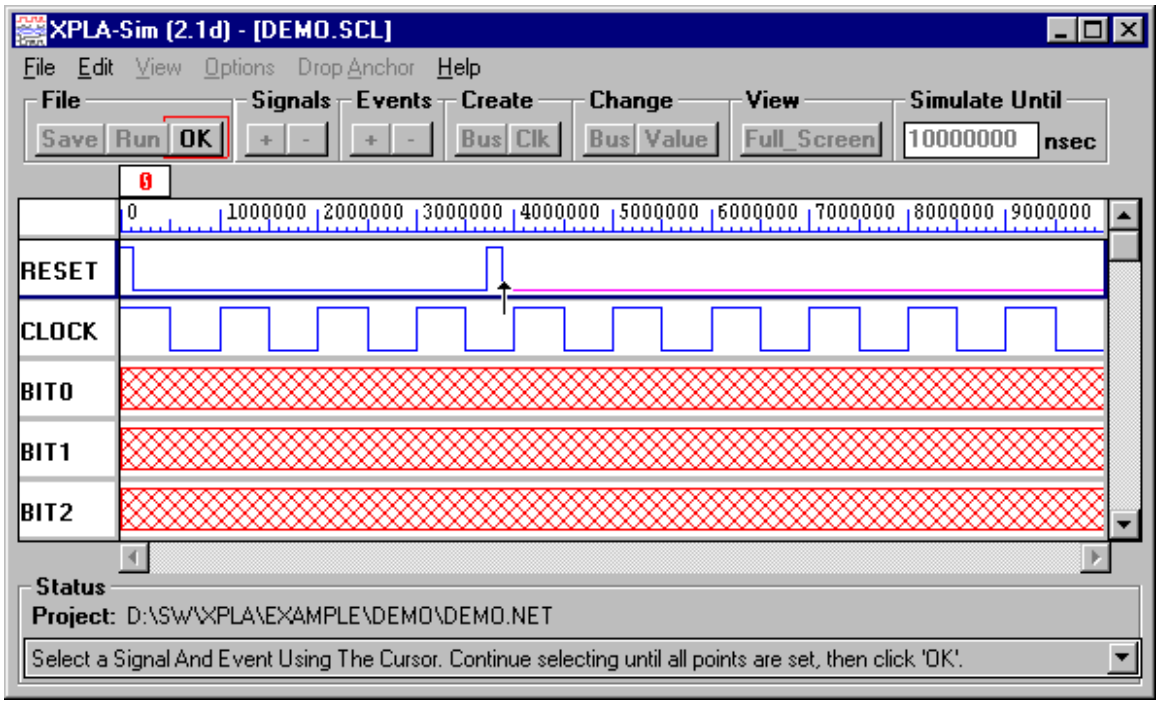


Figure 75

Click on the **OK** button boxed in red to end the add events function. Let's change the predefined clock so that it begins in the '0' state instead of the '1' state. Click on the **Create Clk** button, (for additional help on this function see page 74 under **Add Clk**). The **Clock Settings** window will appear. Click on the **Start at '0'** button and then on the **Accept** button. Now move the cursor to the word **clock** in the signal name row and click on it. Finally, click on the **Done** button inside the **Clock Settings** window. You should have a waveform viewer window very similar to the one shown in Figure 76.

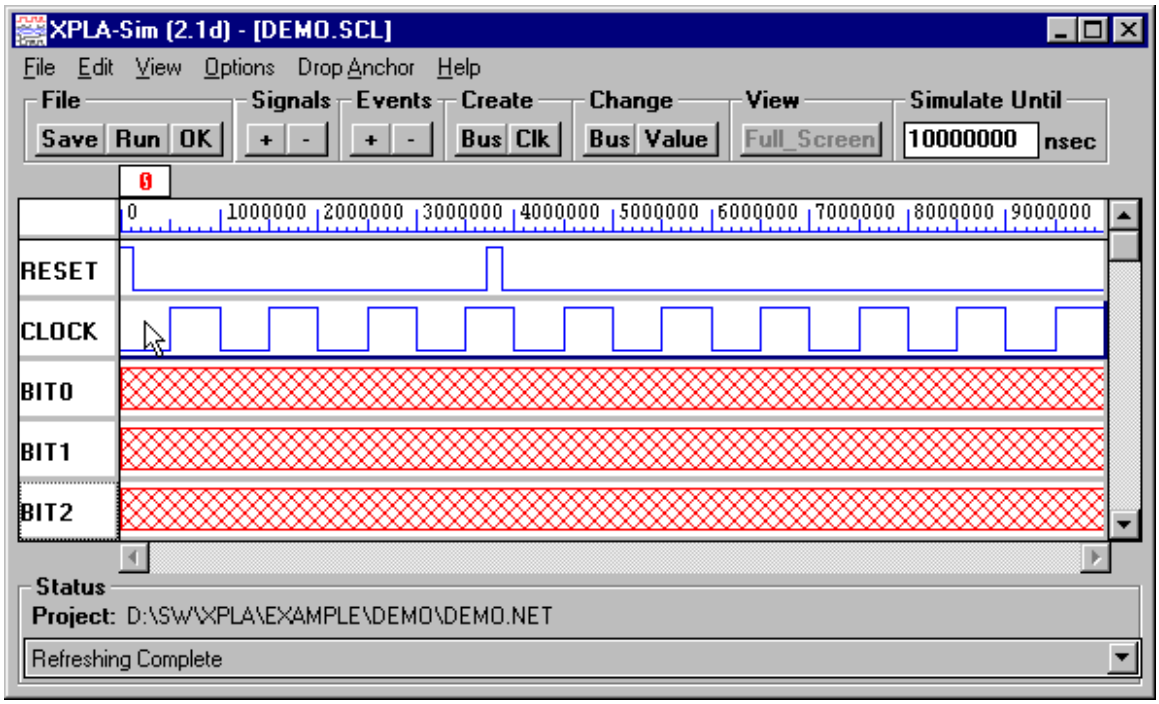


Figure 76

Now click on the run button. The simulator will take the binary design file, combine it with the signal level and timing information contained in the SCL file and present it in graphical form to the waveform viewer window. You will now have a functionally simulated design displayed in the waveform viewer window like the one in Figure 77.

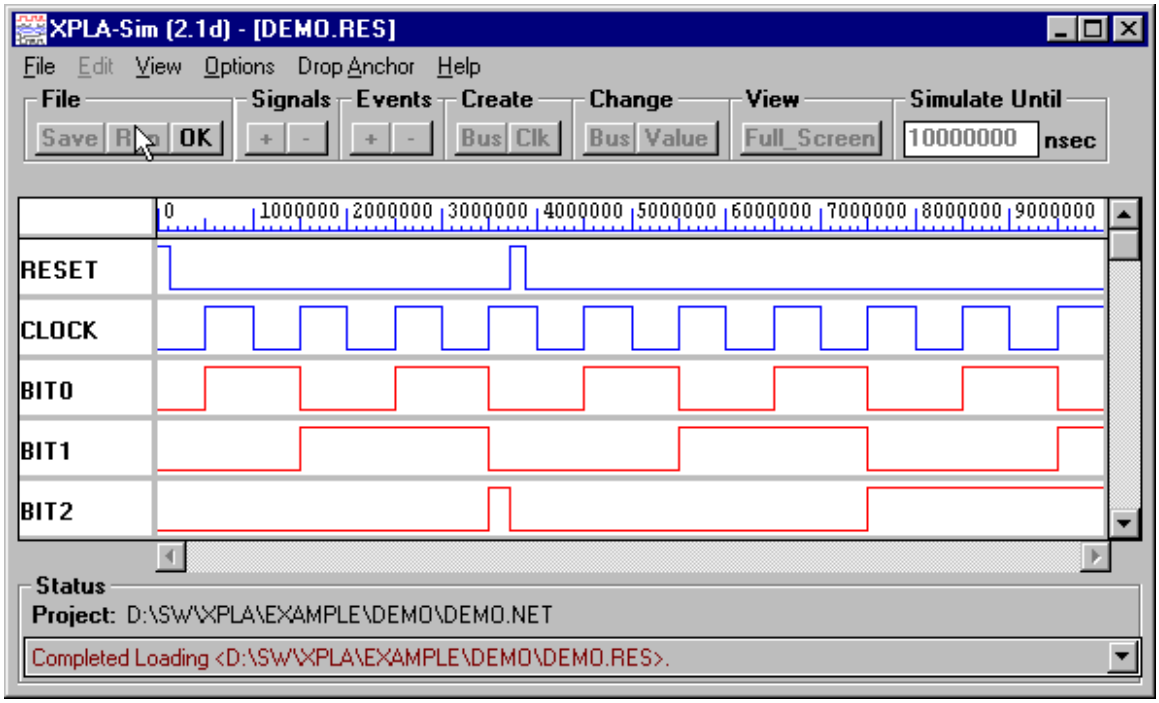


Figure 77

To return the edit mode, click the **OK** button located near the upper left portion of the waveform viewer window. The waveform viewer window will revert to a screen similar to the one shown in Figure 78

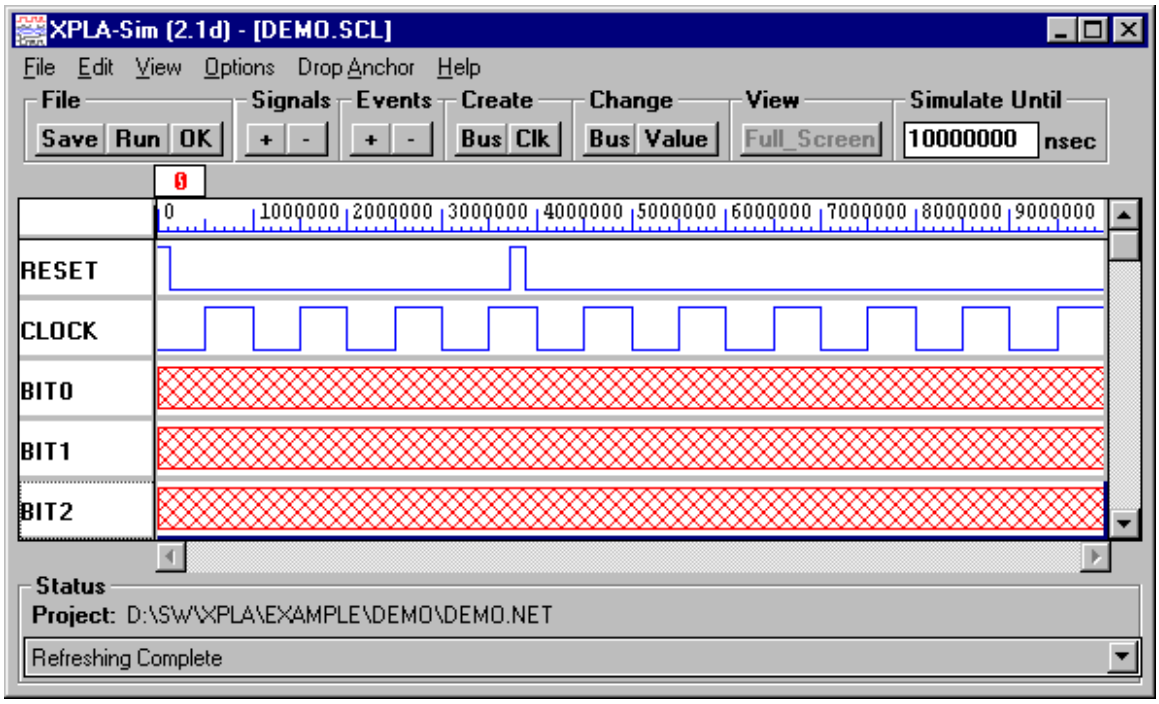


Figure 78

To do a timing simulation, all that is required to be done is to fit the design into a particular part type and package first before running the simulation. The fitter algorithm contains device and package specific timing information that is passed to the simulator. This timing information is incorporated into the signal display when the simulation is run. When you use the anchor feature to measure delays, they will accurately represent those you should expect the actual part to produce.

There are other features in the **XPLA™ Designer Simulator** that have not been mentioned in this overview. Now is the time to play with the simulator and discover the many features it contains. You will find the simulator to be intuitive, fast, powerful, and every easy to use.

- **SCL Brief**

Philips SCL (simulation control language) is divided into three major sections.

1. Data Control Statements

Statements that handle signals, data, and variables.

2. **Flow/Timing**
   Statements that control the simulation flow and timing.
3. **Print Control Statements**
   Statements that control the data to the output file, (**DESIGN.RES**).

Figure 79 and Figure 80 contain a list of **data control statements**, their descriptions and formats.

	Description	Format
IT	Initialize To	IT <init list> (<sign. list>)
ST	Set To	ST <state list> (<sign. list>)
S	Sequence	S init state (t1,t2,..,etc) <list>
INCR	Increment Variable	INCR <variable>, value
IFV	Condition Check Variable	IFV <var><rel><value><SCL>
BUSI	Switch Bus to Input State	BUSI<signal list>
BUSO	Switch Bus to Output State	BUSO <signal list>

Figure 79

	Description	Format
DATS	Start of Data Lines	DATS
DATE	End of Data Lines	DATE
DATV	Ass. Data Lines to Variable	DATV <variable list>
DECV	Decode a Variable	DECV <variable> (<signal list>)
SDC	Set Data Counter	SDC <value>
SETV	Set a Variable to a Value	SETV <variable> <value>
ST DATA	Set to Data	ST DATA <signal list>

Figure 80

Figure 81 and Figure 82 contain a list of **timing control statements**, their descriptions and formats.

	Description	Format
SU	Simulate Until	SU TIME= *+delta
SUNS	Simulate Until Network Stable	SUNS
IF	IF Statement	IF TIME=<ntime> SCL
F	Finish	F
GT	Goto Statement	GT <label>

Figure 81

	Description	Format
CALL	Subroutine Call	CALL <subname>
SUB	Start Subroutine Declaration	SUB <subname>
RET	Return from Subroutine	RET
END	End Subroutine Declaration	END

Figure 82

Figure 83 and Figure 84 contain a list of **print control statements**, their descriptions and formats. ***Print Control Support statement are not supported in this version of the SCL simulator.***

	Description	Format
PT	Print Title	PT 'textstring'
P	Print	P <signal list>
PC	Print Comment	PC 'textstring'
PE	Print Every	PE n1 (n2)
PCO	Print Changes Only	PCO
PV	Print Variables	PV <variable list>

Figure 83

	Description	Format
HP	Halt Printer	HP
SP	Start Printer	SP
LIST	List Signals State "*" or "#"	LIST <specification>
TRAC	Transition Check	TRAC
STAB	Stability Check	STAB <n>

Figure 84

Below is the **SCL** listing for the design **Demo.phd**.

```
*    D:\XPLA\EXAMPLE\DEMO\DEMO.SCL
*    04-23-1996    12:33:49
*    XPLA-Sim (Beta 1.69)
*
*    These Are The Bus Definitions
```

```
DEFBUS QBUS(Q0, Q1, Q2)

*    These Signals Will Be Viewable After Running The Simulator

P RESET, CLOCK, Q0, Q1, Q2, QBUS

*    These Are The Initializations.

IT 01 (GND, VCC)


PCO
*    These Are The Signal Transitions For The Simulation

S 1 (91, 417, 449, 629, 638) RESET
S 1 (17, 34, ETC) CLOCK


SU time = 1000
F
```

Chapter 7 *Controlling the Fitting Process*

Once the design has been compiled and functionally debugged, it can be fit into the targeted device. There are many options available in the XPLA Designer that allow you to control specifically how the design is fitted. These options affect such things as design speed, logic minimization, the width of logic equations, and pin locations. As a general rule, designs that use a large percentage of the available logic in the part will be affected more by the option settings. For some designs, it may be desirable to try different fitting options to obtain optimal results. This chapter describes how to access different fitting options for Philips CPLDs and makes recommendations for settings.

Fitter Options

Fitting options are controlled by customizing the properties of processes to be performed on the source being fitted. For each source, a number of properties may be set to control the compilation and logic reduction; and there are additional options selectable from the XPLA Designer User Interface, as shown in Figure 85, for the targeted device.

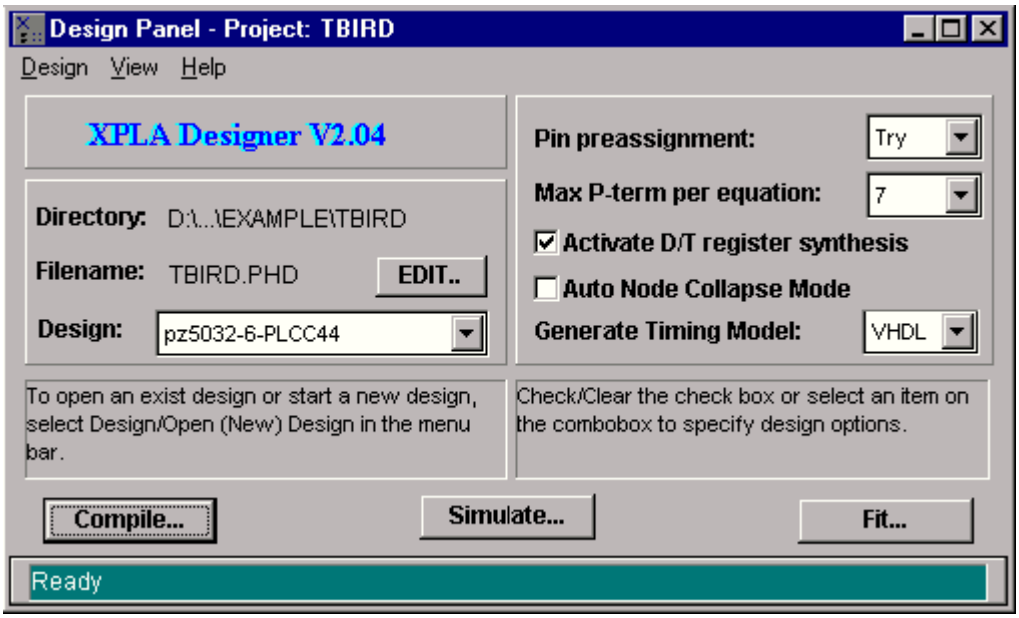

Design Panel - Project: TBIRD
Design View Help
XPLA Designer V2.04
Directory: D:\...\EXAMPLE\TBIRD
Filename: TBIRD.PHD
EDIT..
Design: pz5032-6-PLCC44
Pin preassignment: Try
Max P-term per equation: 7
Activate D/T register synthesis
Auto Node Collapse Mode
Generate Timing Model: VHDL
To open an exist design or start a new design, select Design/Open (New) Design in the menu bar.
Check/Clear the check box or select an item on the combobox to specify design options.
Compile...
Simulate...
Fit...
Ready

Figure 85

The fitting options selectable through the XPLA Designer Interface include:

- Pin Pre-assignments
- Max P-terms per equation
- Activate D/T register synthesis
- Auto Node Collapse Modes

All of these options are described in the next subsections.

Pin Pre-assignments

The pin pre-assignments selection gives the user three different options:

- Try
- Keep
- Ignore

The Try selection will attempt to fit the design with the pin assignments specified in the PHDL source code. If the design cannot be fit with these pin assignments, the fitter will remove the pre-assigned pins and attempt to fit the design with no pre-assigned pins. A warning message will tell the user if the pre-assigned pins have been removed.

The Keep selection will attempt to fit the design with the pin assignments specified in the PHDL source code. If the design cannot be fit with these pin assignments, the fitter will notify the user that the device could not fit. It will not unlock the pins under this option.

The Ignore selection will attempt to fit the design and will ignore the pin assignments specified in the PHDL source code. If the design can be fit with no pre-assigned pins, the fitter will assign pins, which can be viewed in the fitter report (filename.fit). The user should take these pin assignments and incorporate them back into the PHDL source design file. The user will be notified whether the fitting operation was successful.

Max P-terms per equation

The Max P-terms per equation selection allows the user to specify the maximum number of product terms that can be used for each macrocell equation. The reason for this option is to control how wide logic functions are implemented. Wide logic functions can be implemented in two different manners; using more PLA product terms or using more than one pass through the logic array. The maximum number which can be specified ranges from 5 to 37 product terms. As specified in Chapter 3, "XPLA™ Architecture Overview", each macrocell has 5 dedicated PAL product terms and has access to 32 PLA product terms. Because each design is unique and everyone has different system timing requirements, XPLA Designer allows the user control this fitting parameter. Implementing

wide logic functions using 1 or up to 32 of the PLA product terms will add an additional delay (see individual data sheets for PLA delay time). Implementing wide logic functions using multiple passes through the logic array adds a Tpd delay for every additional logic array pass. If speed is the most important parameter, the design should specify a larger value for the "Maximum Product terms per Equation". If fitting the design is perceived to be an issue (density problem) and speed is not as important, then the user should specify a smaller value for the "Maximum Product Terms per Equation".

Activate D/T register synthesis

The Activate D/T register synthesis selection instructs XPLA Designer to minimize the number of product terms using either D or T Type flip-flops. When the selection is activated, the software will implement each equation using all D or all T Type flip-flops and it will pick the flip-flop type that requires the minimum number of product terms. If the number of product terms are the same with either implementation, the XPLA Designer will default to a D-Type flip-flop.

If this selection is not activated, the XPLA Designer will use the type of flip-flop specified in the PHDL source code. If the register type is not specified in the PHDL source code, the XPLA Designer will default to a D-Type flip-flop.

Auto Node Collapse Mode

The Auto Node Collapse Mode is used to minimize the number of passes through the logic array by collapsing internal nodes into other logic. Collapsing the node frees a macrocell at the expense of using more PLA terms, thus maximizing design performance because one pass through the PAL and PLA arrays is faster than multiple passes through the PAL array. When the Auto Node Collapse Mode is used, the compiler will attempt to collapse all internal nodes except those that have the keep attribute specified. When the Auto Node Collapse Mode is not used, the compiler will not attempt to collapse any internal nodes unless the collapse attribute has been specified.

Fitter Option Examples

The following subsections give design examples of how the fitting options can be used to optimize the design to meet the user's individual needs.

Constraining Pins and Nodes Example

One of the features of Philips CPLDs is that you can route virtually 100% of all designs with all pins locked, and all of the macrocells and pins used. This ensures that you can

lock the pins early in the design without having to worry about making changes in the logic that will affect keeping the pinouts fixed. The method to fix the pinouts for a design is to assign a pin number to each input and output in the PHDL source code and specify the **Keep** selection on the compiler options screen.

To lock a pin in a PHDL source, specify the desired pin number in the declarations section. The following example shows how to lock the pins for a dual four bit adder. Pins a1 through a4 will be on pin numbers 5-8, SYNC_CLK will be on pin 9, and so on.

```
MODULE adder
TITLE 'Two four bit adders'
DECLARATIONS
"Inputs
        [a1..a4], [b1..b4]          PIN 5, 6, 7, 8, 11, 12, 13, 14;
        SYNC_CLK                    PIN 9;
        in1, in2                    PIN 1, 2;

"Outputs
        [sync_out1..sync_out4]      PIN 31, 32, 33, 34 ISTYPE 'reg_d, buffer';
        [async_out1..async_out4]    PIN 36, 37, 38, 39 ISTYPE 'reg_d, buffer';
```

To unlock a pin in a PHDL source, remove the pin number that was specified in the declarations section.

D/T Register Synthesis Example

This example show the differences between using D and T Type flip-flops. This is an example of a 16-bit counter targeted for a PZ5032 which is compiled first without "Activate D/T Register Synthesis" selected and then with "Activate D/T Register Synthesis" selected. Figure 86 gives the source code for this 16-bit counter.

```
MODULE count_16

TITLE '16-bit Loadable Counter'

  D15..D0                 pin;
  COUNT15..COUNT0         pin istype 'reg';
  LD, CE, CLK, RST        pin;

  COUNT       = [COUNT15..COUNT0];
  D           = [D15..D0];


EQUATIONS

  when (LD) then COUNT := D;
  else when (CE) then COUNT := COUNT.Q + 1;
  else COUNT := COUNT.Q;
  COUNT.C = CLK;
  COUNT.AR = RST;

END
```

Figure 86

Figure 87 gives the optimized equations for the 16-bit counter without the “Activate D/T Register Synthesis” selected. Please note that only a few output equations are shown to give the reader an idea of the effect on the equation by this option. Also note that, in this case, the number of product terms required by each output increases by one going from Q0 to Q15.

```
COUNT0.D = !LD & CE & !COUNT0.Q
     # !LD & !CE & COUNT0.Q
     # LD & D0;
 COUNT1.D = !LD & CE & COUNT0.Q & !COUNT1.Q
     # !LD & CE & !COUNT0.Q & COUNT1.Q
     # !LD & !CE & COUNT1.Q
     # LD & D1;

                    .
                    .
                    .

COUNT15.D = !LD & CE & COUNT0.Q & COUNT1.Q & COUNT2.Q & COUNT3.Q
& COUNT4.Q
    & COUNT5.Q & COUNT6.Q & COUNT7.Q & COUNT8.Q & COUNT9.Q &
COUNT10.Q &
   COUNT11.Q & COUNT12.Q & COUNT13.Q & COUNT14.Q & !COUNT15.Q
     # !LD & CE & !COUNT14.Q & COUNT15.Q
     # !LD & CE & !COUNT13.Q & COUNT15.Q
     # !LD & CE & !COUNT12.Q & COUNT15.Q
     # !LD & CE & !COUNT11.Q & COUNT15.Q
     # !LD & CE & !COUNT10.Q & COUNT15.Q
     # !LD & CE & !COUNT9.Q & COUNT15.Q
     # !LD & CE & !COUNT8.Q & COUNT15.Q
     # !LD & CE & !COUNT7.Q & COUNT15.Q
     # !LD & CE & !COUNT6.Q & COUNT15.Q
     # !LD & CE & !COUNT5.Q & COUNT15.Q
     # !LD & CE & !COUNT4.Q & COUNT15.Q
     # !LD & CE & !COUNT3.Q & COUNT15.Q
     # !LD & CE & !COUNT2.Q & COUNT15.Q
     # !LD & CE & !COUNT1.Q & COUNT15.Q
     # !LD & CE & !COUNT0.Q & COUNT15.Q
     # !LD & !CE & COUNT15.Q
     # LD & D15;
```

Figure 87

Figure 88 gives the fitting report when the “Activate D/T Register Synthesis” option is not selected.

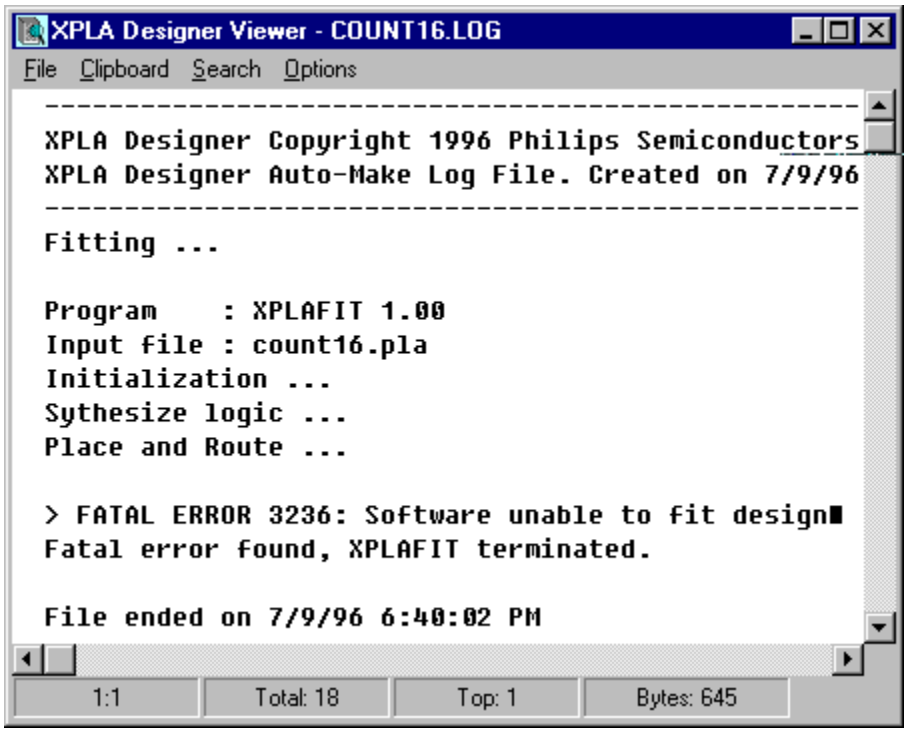

XPLA Designer Viewer - COUNT16.LOG
File Clipboard Search Options
XPLA Designer Copyright 1996 Philips Semiconductors
XPLA Designer Auto-Make Log File. Created on 7/9/96
Fitting ...
Program : XPLAFIT 1.00
Input file : count16.pla
Initialization ...
Sythesize logic ...
Place and Route ...
> FATAL ERROR 3236: Software unable to fit design
Fatal error found, XPLAFIT terminated.
File ended on 7/9/96 6:40:02 PM
1:1
Total: 18
Top: 1
Bytes: 645

Figure 88

As one can see from Figure 88, the device did not fit without the "Activate D/T Register Synthesis" selected.

Figure 89 gives the optimized equations for the 16-bit counter with the "Activate D/T Register Synthesis" selected. Please note that only a few output equations are shown to give the reader an idea of the effect on the equation by this option. Also note that, in this case, the number of product terms required by each output stays constant.

```
COUNT0.D = !LD & CE & !COUNT0.Q
    # !LD & !CE & COUNT0.Q
    # LD & D0;
 COUNT1.T = LD & !D1 & COUNT1.Q
    # LD & D1 & !COUNT1.Q
    # !LD & CE & COUNT0.Q;
        .
        .
        .
COUNT15.T = LD & !D15 & COUNT15.Q
    # LD & D15 & !COUNT15.Q
```

```
    # !LD & CE & COUNT0.Q & COUNT1.Q & COUNT2.Q & COUNT3.Q &
COUNT4.Q &  COUNT5.Q & COUNT6.Q & COUNT7.Q & COUNT8.Q &
COUNT9.Q & COUNT10.Q & COUNT11.Q & COUNT12.Q & COUNT13.Q &
COUNT14.Q;
END
```

Figure 89

Figure 90 gives the fitting report when the "Activate D/T Register Synthesis" option is selected.[4]

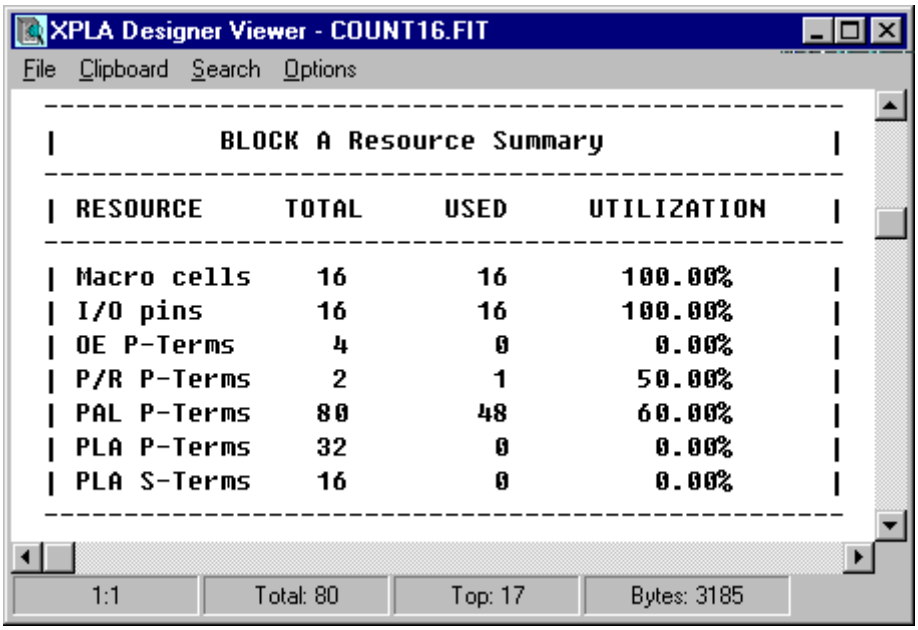


Figure 90

As one can see from Figure 90, the device did fit with the "Activate D/T Register Synthesis" selected. In fact, only Logic Block A was used to implement this 16-bit counter. Logic Block B is still available to implement other functions.

Device Utilization Verses Design Speed Example

One of the key settings that can most affect the fitting of your design is the **Max P-Term per equation** selection. This property determines the maximum number of product terms

[4] *PLEASE NOTE: Some Quantum Physics Theories Suggest That When the Consumer Is Not Directly Observing This Product, It May Cease to Exist or Will Exist Only in a Vague and Undetermined State.*

that may be summed for any macrocell. There is a trade-off between design speed and efficiency of device utilization that must be considered when setting this property. When the logic equation for a macrocell has a number of product terms that exceeds the maximum count selected, buried nodes will be used to break the equation into smaller parts that will then be summed in another macrocell, reducing the maximum frequency at which the design can run. As an example, consider the equation:

```
OUT := A # B # C # D # E # F # G # H
```

and assume that the P-term maximum is set to 6. Because the equation has more than 6 product terms, it cannot be implemented in one macrocell. After fitting, the equation would be broken into two parts and implemented something like:

```
BURIED_NODE = A # B # C # D # E;
OUT := BURIED_NODE # F # G # H;
```

which would now require two macrocells and would slow down the speed at which the design would be able to run due to the fact that resolution of signal OUT now requires two passes through the logic array. For each stage of buried nodes that would need to be created to meet the P-term limit set, you must add a T_{pd} to the T_{su} of the final output for timing. If the P-term limit were changed to 8, no internal node will be created and the design will run at a faster speed because only one pass through the logic array is required to resolve the signal OUT.

Now consider the case where a large percentage of the macrocells require large sums of product terms. Because of the way the fitter allocates the product terms, the P-term limit setting will determine if the design will fit in the part. When fitting the design, the first thing the Philips fitter does is create sums of products (logic equations) based on the maximum width specified in the P-term limit property. The first five product terms for each output are implemented in the dedicated terms of the PAL array, and the remaining terms are taken from the PLA array. Each product sum is implemented by combining those PAL and PLA terms. If all of the PLA terms are used before all of the wide product sums have been allocated, the design will not fit in the device. For example, consider a 32 macrocell device with 16 outputs of 10 product terms each. Between the two logic blocks, there are 64 PLA product terms available. If the P-term limit were set to 10, the fitter would first create the 20 equations with the required 10 product terms each (no internal nodes) because they all fall under the P-term limit. When it started allocating the logic, the fitter would use five PAL terms and five PLA terms for each output. A simple calculation shows that it would require 16 * 5 = 80 PLA product terms to do the design this way. This exceeds the available 64 PLA product terms and therefore the design will not fit. On the other hand, if the P-term limit were set to 6, the fitter would build a buried node for each output that comprised 6 of the 10 required product terms. The buried node would then be combined on a second pass through the array with the remaining four product terms for the final result. This limit would use all 32 of the available macrocells,

but only 16 PLA product terms. The design would now fit in the device, but the maximum frequency would be reduced due to the second pass through the logic array.

For Philips devices, it is recommended that you start your design with a P-term limit of 7. This is the best starting point for balancing logic width and speed with device utilization. The number can then be adjusted up or down depending on specific design requirements.

Another aspect of device fitting closely related to the trade-off between speed and device utilization illustrated above is the ability to make changes late in the design without having to change the existing pinout. Philips devices have excellent pinout retention characteristics due to the dedicated PAL product terms plus the extra PLA product terms through which late design changes can be routed to any of the existing macrocells.

Chapter 8 *PHDL Language Reference*

The following chapter provides detailed syntax and usage descriptions for all PHDL commands. The commands are listed in alphabetical order. For each command, there is a description, a general syntax listing, and an example. Expressions enclosed within brackets [.....] are optional.

case

Description

The case statement is used within state diagrams to resolve state transitions based on one or more conditions.

Syntax

case (*condition*): *state*;
    [(*condition*): *state*;]
    [(*condition*): *state*;]
        .
        .
        .
    [(*condition*): *state*;]
endcase;

Condition is a logical expression that, when true, causes the state diagram to transition to the corresponding *state*.

Example

```
Module  case_ex
Title   'Example of case statement'

in1,in2,in3     pin;
out1,out2       pin istype 'com,buffer';
q1,q0           node istype 'reg';

state_diagram [q1,q0]
    state [0,0]:    out1 = 0;
                    out2 = 0;
                    case (in1 == 1) & (in2 == 1): [0,1];
                        (in1 == 1) & (in2 == 0): [1,1];
```

```
                    endcase;
        state [0,1]:  out1 = 0;
                      out2 = 1;
                      case (in1 == 0): [1,0];
                      endcase;
        state [1,0]:  out1 = 1;
                      out2 = 0;
                      case (in2 == 0): [0,0];
                          (in1 == 1) & (in2 == 1): [1,1];
                      endcase;
        state [1,1]:  out1 = 1;
                      out2 = 1;
                      case (in2 == in1): [0,0];
                      endcase;

end
```

declarations

Description

The declarations statement is an optional word that may be placed near the beginning of the PHDL source file to indicate the start of the declarations section. The declarations section is where constants, variables, macro functions, and signals are declared. Refer to chapter 5 for more information.

Syntax

declarations
    [*constants*;
    variables;
    macro functions;
    signals;]

The constants, variables, macro functions, and signals are all optional, but any source will have at least one of these.

Example

```
Module  declar
Title   'Example of declarations'

declarations
```

```
a,b,c,d          pin;
out              pin istype 'com,invert';

equations

out = a & b & c # d;

end
```

else

Description

The keyword else is part of both the if-then-else structure and the when-then-else structure. It is used to specify another result that should occur when conditions specified by the if or when statements are false. For more information, refer to the if-then-else and when-then-else commands in this chapter.

Syntax

```
if (condition) then
        {state};
else
        {state};
```

OR

```
        when (condition) then
                {result};
        else
                {result};
```

For both cases, whenever the logical expression *condition* is false, the *state* or *result* after the else statement will be implemented.

Example

Else used with if statement in a state diagram:

```
Module  if_else
Title   'Example of else with if statement in state diagram'
```

```
in1,in2          pin;
clk              pin;
out              pin istype 'reg';
s1,s2            node istype 'reg';

equations

out.c = clk;
s1.c = clk;
s2.c = clk;

state_diagram [s1,s2]
        state [0,0]:    out = 0;
                        if (in1 == 0) & (in2 == 1) then [0,1]
                        else [1,0];
        state [0,1]:    out = 1;
                        if (in1 == 1) & (in2 == 0) then [1,0]
                        else [1,1];
        state [1,0]:    out = 1;
                        if (in1 == 1) & (in2 == 1) then [1,1]
                        else [0,0];
        state [1,1]:    out = 0;
                        if (in1 == 0) & (in2 == 0) then [0,0]
                        else [0,1];


end
```

Example

Else used with when statement in equations:

```
Module  when_else
Title   'Example of else with when statement in equations'

in1,in2          pin;
clk              pin;
out1,out2        pin istype 'reg';

equations

out1.c = clk;
out2.c = clk;
```

```
when (in1 == 1) & (in2 == 1) then out1 = 1;
else out1 = 0;

when (in1 == 0) & (in2 == 0) then out2 = 0;
else out2 = 1;

end
```

end

Description

The reserved word end is required to be the last word in every PHDL module. It is the counter part to the module statement.

Syntax

module
header;
declarations;
logic description;
end

The header, declarations, and logic description make up the PHDL design. Refer to chapter 5 for more information.

Example

```
Module  end_ex
Title   'Very simple example showing use of end statement'

out      pin istype 'com';

equations

out = 1;

end
```

endcase

Description

The reserved word endcase is used to indicate the end of a case statement. Refer to case in this chapter for more information.

Syntax

case (*condition*): *state*;
[(*condition*): *state*;]
[(*condition*): *state*;]
.
.
.
[(*condition*): *state*;]
endcase;

Example

```
Module  case_ex
Title   'Example of endcase statement'

in1,in2         pin;
out1,out2       pin istype 'com,buffer';
q0              node istype 'reg';

state_diagram [q0]
        state [0]:      out1 = 1;
                        out2 = 0;
                        case (in1 == 0) & (in2 == 1): [1];
                        endcase;
        state [1]:      out1 = 0;
                        out2 = 1;
                        case (in1 == 1): [0];
                        endcase;

end
```

equations

Description

The reserved word equations is used to indicate the start of one or more logic equations. It is used in the logic description section of the PHDL file.

Syntax

equations
element = *expression*;
element := *expression*;
.
.
.
when-then-else;

Note that there are two different types of assignment operators. When signals are specified by including attributes in the pin or node statements of the declarations section, the two different assignment types are functionally equivalent. When signals are not specified with attributes, and are only indicated to be pins or nodes, then the different types of assignments have different effects on the design. When the single equal sign (=) assignment is used, the signal is assumed by the compiler to be a combinatorial signal; when the colon-equal (:=) assignment is used, the signal is assumed to be a registered signal. It is recommended for design clarity that you specify all signals with attributes, in which case the assignment operators are equivalent.

Example

```
Module  equat
Title   'Example of reserved word equations'

a,b,c,d         pin;
clk             pin;
out1,out2       pin istype 'com';
out3,out4       pin istype 'reg';

equations

out3.c = clk;
out4.c = clk;

out1 = a & b $ c & d;
out2 := !a # !b # c # d;
```

```
when (a == 1) then out3.d = b & c;
else out3.d = out3.q;

out4 = out1.com & a # !b;

end
```

goto

Description

The goto statement is used in a state diagram to force a state transition to a specified state. Goto is often used as in the example below to transition out of unspecified states.

Syntax

goto *specified_state*;

Specified_state is the specified state for the unconditional transition.

Example

```
Module  goto_ex
Title   'Example of state transition with goto statement'

in      pin;
clk     pin;
out     pin istype 'com';
s1,s0   node istype 'reg';

equations

s0.c = clk;
s1.c = clk;

state_diagram [s1,s0]
        state [0,0]:    out = 0;
                        if (in == 1) then [0,1];
        state [0,1]:    out = 1;
                        if (in == 1) then [1,0];
        state [1,0]:    out = in;
                        if (in == 1) then [0,0];
```

```
        state [1,1]:    goto [0,0];

end
```

if-then-else

Description

The if-then structure is used in state diagrams to specify conditional state transitions. **This structure cannot be used for equations; it is only used in state diagrams.**

Syntax

if (*condition*) then
        {*state*};
[else if (*condition*) then
                {*state*};]
[else
        {*state*};]

When a *condition* is true, the state diagram will transition to the corresponding *state*. If none of the *condition*s are true, the state diagram will transition to the *state* specified after the final else statement. Note that both of the else structures are optional. If-then can be used by itself, with a single else, or with multiple levels of else-if statements. If-then-else structures can also be nested. Refer to the examples below.

Example

Simple if-then structure

```
Module  ifthen
Title   'Example of simple if-then structure'

in              pin;
clk             pin;
out             pin istype 'reg';
s1              node istype 'reg';

equations

out.c = clk;
s1.c = clk;
```

```
state_diagram [s1]
        state [0]:      out = 0;
                        if (in == 1) then [1];
        state [1]:      out = 1;
                        if (in == 1) then [0];

end
```

Example

If-then-else structure

```
Module  iftelse
Title   'Example of else with if statement in state diagram'

in1,in2         pin;
clk             pin;
out             pin istype 'reg';
s1,s2           node istype 'reg';

equations

out.c = clk;
s1.c = clk;
s2.c = clk;

state_diagram [s1,s2]
        state [0,0]:    out = 0;
                        if (in1 == 0) & (in2 == 1) then [0,1]
                        else [1,0];
        state [0,1]:    out = 1;
                        if (in1 == 1) & (in2 == 0) then [1,0]
                        else [1,1];
        state [1,0]:    out = 1;
                        if (in1 == 1) & (in2 == 1) then [1,1]
                        else [0,0];
        state [1,1]:    out = 0;
                        if (in1 == 0) & (in2 == 0) then [0,0]
                        else [0,1];


end
```

Example

Multiple if-then-else structure

```
Module  mult_ite
Title   'Example of multiple if-then-else structure'

in1,in2         pin;
clk             pin;
out             pin istype 'reg';
s1,s2           node istype 'reg';

equations

out.c = clk;
s1.c = clk;
s2.c = clk;

state_diagram [s1,s2]
        state [0,0]:    out = 0;
                        if (in1 == 0) & (in2 == 1) then [0,1]
                        else if (in1 == 1) & (in2 == 0) then [1,0]
                        else if (in1 == 1) & (in2 == 1) then [1,1]
                        else [0,0];
        state [0,1]:    out = 1;
                        if (in1 == 0) & (in2 == 0) then [0,0]
                        else if (in1 == 1) & (in2 == 0) then [1,0]
                        else if (in1 == 1) & (in2 == 1) then [1,1]
                        else [0,1];
        state [1,0]:    out = 1;
                        if (in1 == 0) & (in2 == 0) then [0,0]
                        else if (in1 == 0) & (in2 == 1) then [0,1]
                        else if (in1 == 1) & (in2 == 1) then [1,1]
                        else [1,0];
        state [1,1]:    out = 0;
                        if (in1 == 0) & (in2 == 0) then [0,0]
                        else if (in1 == 0) & (in2 == 1) then [0,1]
                        else if (in1 == 1) & (in2 == 0) then [1,0]
                        else [1,1];

end
```

Example

Nested if-then-else structure

```
Module  nested
Title   'Example of nested if-then-else statement in state diagram'

in1,in2         pin;
clk             pin;
out             pin istype 'reg';
s1,s2           node istype 'reg';

equations

out.c = clk;
s1.c = clk;
s2.c = clk;

state_diagram [s1,s2]
        state [0,0]:    out = 0;
                        if (in1 == 0) then
                                {
                                if (in2 == 0) then [1,1]
                                else [0,1];
                                }
                        else [1,0];
        state [0,1]:    out = 1;
                        if (in1 == 0) then
                                {
                                if (in2 == 0) then [0,0]
                                else [1,0];
                                }
                        else [1,1];
        state [1,0]:    out = 1;
                        if (in1 == 1) then
                                {
                                if (in2 == 0) then [0,1]
                                else [1,1];
                                }
                        else [0,0];
        state [1,1]:    out = 0;
                        if (in1 == 0) then
                                {
                                if (in2 == 0) then [1,0]
                                else [0,0];
                                }
                        else [0,1];
```

```
end
```

istype

Description

The istype statement is used to attach attributes to pins and nodes in a PHDL design. The attributes that are attached determine the format of the output signal attached to the pin or node. The table below lists all available attributes associated with the istype statement and describes their function. For more information, refer to the section on signals in chapter 5.

ATTRIBUTE	DESCRIPTION
buffer	**Use non-inverted output.**
collapse	**Collapse this signal during logic synthesis.**
com	**Combinatorial output.**
invert[5]	**Use inverted output.**
keep	**Do not collapse this signal during logic synthesis.**
reg	**Clocked memory element.**
reg_d	**D type flip-flop clocked memory element.**
reg_g	**Gated D type flip-flop clocked memory element.**
reg_jk	**JK type flip-flop clocked memory element.**
reg_sr	**SR type flip-flop clocked memory element.**
reg_t	**T type flip-flop clocked memory element.**
retain	**Do not minimize this output. Preserve redundant product terms.**

Syntax

signal_name pin istype '*attributes*';

OR

signal_name node istype '*attributes*';

Signal_name is the name assigned to the signal associated with the pin or node, and *attributes* are from the list in the above table. Multiple attributes may be assigned at one time if they are separated by commas. Multiple attributes must not conflict with each

[5] This attribute is ignored for all signals except those that are a detailed register type (reg_d, reg_g, reg_jk, reg_sr, reg_t). If you have other signals that are active low, you must build the logic equations accordingly.

other; i.e. a pin or node cannot simultaneously be assigned the attributes collapse and keep.

Example

```
Module  istyp_ex
Title   'Example showing istype statement usage'

"Declare 4 inputs and a clock

a,b,c,d         pin;
clk             pin;

"Declare 2 combinatorial outputs that are not to be collapsed

out1,out2       pin istype 'com,keep';

"declare 2 registered outputs that are to be inverted output

out3,out4       pin istype 'reg,invert';

"declare 2 d-type registered nodes

n1,n2           node istype 'reg_d';


equations

out3.c = clk;
out4.c = clk;
n1.c = clk;
n2.c = clk;

out1 = a & b;
out2 = c # d & n1.q;
out3 := a # b # out1.com;
out4 := c & d & !a;
n1 := out1.com & out2.com;
n2 := out3.q & out4.q;

end
```

macro

Description

The reserved word macro is used to declare macro functions in the declarations section of the PHDL file. A macro function is a set of PHDL code that can be used throughout source file multiple times without having to retype the code each time. For more information, refer to the section titled “macro functions” in chapter 5.

Syntax

macro_name macro (*inputs*, *outputs*)
{
macro definition;
};

Macro_name is the name of the macro function being defined, *inputs* and *outputs* are the variables passed into and out of the function, and *macro definition* is where the relationship between the inputs and outputs is created. Note in the example below how question marks are used in front of the inputs and outputs in the macro definition. These question marks are required for the macro to be instantiated in the PHDL file several times, each time with different inputs and outputs.

Example

```
Module  macro_ex
Title   'Simple macro function example'

a,b,c,d,e,f,g     pin;
o6..o1  pin istype 'com';

"Create the macro called lgc_dsn

lgc_dsn macro (in1,in2,in3,in4,out1,out2)
        {
        ?out1 = ?in1 & ?in2 & ?in3;
        ?out2 = !?in1 # ?in4 & ?in2;
        };


" NOTE reserved word equations is required due to equations in the macro


EQUATIONS
```

```
" Use the macro in the design

lgc_dsn (a,b,d,e,o6,o5);
lgc_dsn (d,e,f,g,o1,o3);
lgc_dsn (a,c,e,g,o2,o4);

end
```

module

Description

The reserved word module is used to assign a name to the PHDL design. Module and a name for the design are required as the first line in every PHDL source.

Syntax

module *module_name*

Example

```
Module  modl_ex
Title   'Example of the reserved word module'

in      pin;
out     pin istype 'com';

equations

out = in;

end
```

node

Description

The reserved word node is used to declare a signal as type node. Refer to the signals section in chapter 5 for more information on nodes.

Syntax

signal_name node istype '*attributes*';

Signal_name is the name assigned to the signal, and *attributes* are identifiers that determine the type of signal. Refer to the istype description for more information.

Example

```
Module  node_ex
Title   'Example of node use'

in1,in2 pin;
clk     pin;
out     pin istype 'com';
n1,n2   node istype 'reg';

equations

n1.c = clk;
n2.c = clk;

state_diagram [n1,n2]
        state [0,0]:    out = 0;
                        if (in1 == 1) then [0,1];
        state [0,1]:    out = in2;
                        if (in1 == 0) then [1,0];
        state [1,0]:    out = 1;
                        if (in2 == 1) then [1,1];
        state [1,1]:    out = in1;
                        if (in2 == 0) then [0,0];

end
```

pin

Description

The reserved word pin is used to declare a signal as type pin. Refer to the signals section in chapter 5 for more information on pins.

Syntax

signal_name pin istype '*attributes*';

Signal_name is the name assigned to the signal, and *attributes* are identifiers that determine the type of signal. Refer to the istype description for more information.

Example

```
Module  pin_ex
Title   'Example of pin command'

clk     pin;
a,b,c   pin;
d,e     pin istype 'com';
f       pin istype 'reg';

equations

f.c = clk;

d = a & b & c;
e = a # b # c;
f := a & c # b & c;

end
```

property

Description

The property statement specifies information about the design to external programmers like the compiler. Property statements are usually included at the beginning of the PHDL file. The following table lists available properties and gives a brief description.

PROPERTY	DESCRIPTION
dut on	**Enables global tri-state function.**

Syntax

XPLA property '*property*';

Example

```
Module  prop
Title   'Example of property statement'
```

```
XPLA property 'dut on';

in      pin;
out     pin istype 'com';

equations

out = in;

end
```

state

Description

The reserved word state is used within a state diagram to indicate the state that is being defined. For more information, refer to the state_diagram section of this chapter and to the logic description section of chapter 5.

Syntax

```
state_diagram [q1,q0]
        state [0,0]:
                state description;
        state [0,1]:
                state description;
        .
        .
        .
        state [1,1]:
                state description;
```

Example

```
Module  stat_ex
Title   'Example showing reserved word state'

in1,in2         pin;
clk             pin;
out             pin istype 'reg';
s1,s2,s3        node istype 'reg';

equations
```

```
out.c = clk;
s1.c = clk;
s2.c = clk;
s3.c = clk;

state_diagram [s1,s2,s3]
        state [0,0,0]:  out = 0;
                        if (in1 == 1) then [0,0,1];
        state [0,0,1]:  out = 1;
                        if (in1 == 1) then [0,1,0];
        state [0,1,0]:  out = in1;
                        if (in1 == 1) then [0,1,1];
        state [0,1,1]:  out = !in1;
                        if (in1 == 1) then [1,0,0];
        state [1,0,0]:  out = in2;
                        if (in1 == 1) then [1,0,1];
        state [1,0,1]:  out = !in2;
                        if (in1 == 1) then [0,0,0];
        state [1,1,0]:  goto [0,0,0];
        state [1,1,1]:  goto [0,0,0];

end
```

state_diagram

Description

The phrase state_diagram is used in PHDL to indicate the start of a state machine design definition. For more information, refer to the state and if-then-else sections of this chapter and to the logic description section of chapter 5.

Syntax

```
state_diagram [q1,q0]
        state [0,0]:
                state description;
        state [0,1]:
                state description;
        .
        .
        .
        state [1,1]:
                state description;
```

State indicates the state that is being defined and state description is where the outputs and state transition logic are created. Note that state_diagrams must use sequential registers in PHDL for representation of the different states. The state transition logic must use the goto or if-then-else structure.

Example

```
Module  diagram
Title   'Example showing state diagram'

in1,in2         pin;
clk             pin;
out             pin istype 'reg';
s1,s2,s3        node istype 'reg';

equations

out.c = clk;
s1.c = clk;
s2.c = clk;
s3.c = clk;

state_diagram [s1,s2,s3]
        state [0,0,0]:  out = 0;
                        if (in1 == 1) then [0,0,1];
        state [0,0,1]:  out = 1;
                        if (in1 == 1) then [0,1,0];
        state [0,1,0]:  out = in1;
                        if (in1 == 1) then [0,1,1];
        state [0,1,1]:  out = !in1;
                        if (in1 == 1) then [1,0,0];
        state [1,0,0]:  out = in2;
                        if (in1 == 1) then [1,0,1];
        state [1,0,1]:  out = !in2;
                        if (in1 == 1) then [0,0,0];
        state [1,1,0]:  goto [0,0,0];
        state [1,1,1]:  goto [0,0,0];

end
```

then

Description

The keyword then is part of both the if-then-else structure and the when-then-else structure. It is used to specify the result that should occur when conditions specified by the corresponding if statement are true. For more information, refer to the if-then-else and when-then-else commands in this chapter.

Syntax

```
if (condition) then
        {state};
else
        {state};
```

OR

```
        when (condition) then
                {result};
        else
                {result};
```

For both cases, whenever the logical expression *condition* is true, the *state* or *result* after the then statement will be implemented.

Example

Else used with if statement in a state diagram:

```
Module  if_then
Title   'Example of then with if statement in state diagram'

in1,in2         pin;
clk             pin;
out             pin istype 'reg';
s1,s2           node istype 'reg';

equations

out.c = clk;
s1.c = clk;
s2.c = clk;

state_diagram [s1,s2]
        state [0,0]:    out = 0;
                        if (in1 == 0) & (in2 == 1) then [0,1]
                        else [1,0];
```

```
        state [0,1]:    out = 1;
                        if (in1 == 1) & (in2 == 0) then [1,0]
                        else [1,1];
        state [1,0]:    out = 1;
                        if (in1 == 1) & (in2 == 1) then [1,1]
                        else [0,0];
        state [1,1]:    out = 0;
                        if (in1 == 0) & (in2 == 0) then [0,0]
                        else [0,1];


end
```

Example

Else used with when statement in equations:

```
Module  when_then
Title   'Example of then with when statement in equations'

in1,in2         pin;
clk             pin;
out1,out2       pin istype 'reg';

equations

out1.c = clk;
out2.c = clk;

when (in1 == 1) & (in2 == 1) then out1 = 1;
else out1 = 0;

when (in1 == 0) & (in2 == 0) then out2 = 0;
else out2 = 1;

end
```

title

Description

The optional title statement is used as a comment at the beginning of a PHDL file to describe the function of the design.

Syntax

title '*what you want the title to be*'

Example

```
Module  simple
Title   'A very simple design with a title'

out pin;

equations

out = 1;

end
```

when-then-else

Description

The when-then-else structure is used with equations to specify conditional results. **This structure cannot be used for state diagrams; it is only used with equations.**

Syntax

```
when (condition) then
        {result};
[else when (condition) then
                {result};]
[else
        {result};]
```

When a *condition* is true, the corresponding *state* result will be implemented. If none of the *condition*s are true, the result specified after the final else statement will be implemented. Note that both of the else structures are optional. When-then-else can be used by itself, with a single else, or with multiple levels of else-when statements. When-then-else structures can also be nested. Refer to the examples below.

Example

Simple when-then structure

```
Module  whenthen
Title   'Example of simple when-then statement in equations'

in1,in2         pin;
clk             pin;
out1            pin istype 'reg';

equations

out1.c = clk;

when (in1 == 1) & (in2 == 1) then out1 = 1;

when (in1 == 0) & (in2 == 0) then out1 = 0;

end
```

Example

When-then-else structure

```
Module  whenthen
Title   'Example of when-then-else statement in equations'

in1,in2         pin;
clk             pin;
out1,out2       pin istype 'reg';

equations

out1.c = clk;
out2.c = clk;

when (in1 == 1) & (in2 == 1) then out1 = 1;
else out1 = 0;

when (in1 == 0) & (in2 == 0) then out2 = 0;
else out2 = 1;

end
```

Example

Multiple when-then-else structure

```
Module  multwhen
Title   'Example of multiple when-then-else'

in1,in2         pin;
clk             pin;
out1,out2       pin istype 'reg';

equations

out1.c = clk;
out2.c = clk;

when (in1 == 1) & (in2 == 1) then
        {
        out1 = 1;
        out2 = 1;
        }
else when (in1 == 1 ) & (in2 == 0) then
        {
        out1 = 0;
        out2 = 1;
        }
else
        {
        out1 = 0;
        out2 = 0;
        }

end
```

Example

Nested when-then-else structure

```
Module  nestwhen
Title   'Example of nested when-then-else'

in1,in2         pin;
clk             pin;
out1,out2       pin istype 'reg';

equations
```

```
out1.c = clk;
out2.c = clk;

when (in1 == 0) then
        {
        when (in2 == 0) then
                {
                out1 = 1;
                out2 = 0;
                }
        else
                {
                out1 = 1;
                out2 = 1;
                }
        }
else when (in1 == 1 ) then
        {
        when (in2 == 0) then
                {
                out1 = 0;
                out2 = 0;
                }
        else
                {
                out1 = 0;
                out2 = 1;
                }
        }

end
```

with

Description

The with statement is used in conjunction with if-then-else and case structures in state diagrams to combine output equations with state transitions. It is also useful to make registered outputs transition with the state, rather than one clock cycle after the state change. Normally, when state diagrams are written without using the with statement, registered outputs change one clock cycle after the state. For more information, refer to the state_diagram, if-then-else, and case sections in this chapter.

Syntax

if (*condition*) then *state* with
{
output equations;
}
else if (*condition*) then *state* with
{
output equations;
}
else *state* with
{
output equations;
}
OR

case (*condition*): *state* with
{
output equations;
}
[(*condition*): *state* with
{
output equations;
}]
.
.
.
[(*condition*): *state* with
{
output equations;
}]
endcase;

Example

With statement in if-then structure

```
Module  if_with
Title   'Example of with statement in if-then structure'

in              pin;
clk             pin;
out             pin istype 'reg';
s1              node istype 'reg';

equations
```

```
out.c = clk;
s1.c = clk;

state_diagram [s1]
        state [0]:      if (in == 1) then [1] with out = 0;
        state [1]:      if (in == 1) then [0] with out = 1;

end
```

Example

With statement in case structure

```
Module  casewith
Title   'Example of with statement in case structure'


in1,in2         pin;
out1,out2       pin istype 'com,buffer';
q0              node istype 'reg';

state_diagram [q0]
        state [0]:      case (in1 == 0) & (in2 == 1): [1] with
                                {
                                out1 = 1;
                                out2 = 0;
                                }
                        endcase;
        state [1]:      case (in1 == 1): [0] with
                                {
                                out1 = 0;
                                out2 = 1;
                                }
                        endcase;

end
```

APPENDIX A *SCL: Simulation Control Language*

The XPLA Designer logic simulator models the operation of a logic design so that the modeled circuit can be analyzed and debugged. The simulator is a vital part of the design process because is it difficult or impossible to analyze the internal operation of an integrated circuit in any other way. The XPLA Designer simulator is similar in its basic operation to most commercially developed simulators.

The simulator runs a simulation of the logic of the design based on the binary netlist (.BIN file) created by the simulator and the input stimulus file (.SCL file) created using a text or waveform editor. The simulator determines the values of the output signals as a function of time, given the input signals in the SCL file.

Theory of Operation

The XPLA Designer simulator is an event-driven 5-State simulator. Event driven means the simulator treats the clock as an ordinary event and advances time (i.e., expands or contracts) according to activity within the circuit (i.e., output transitions on gates). If nothing is happening, the simulated model of "time" is advanced and updated to when there is some *action.*

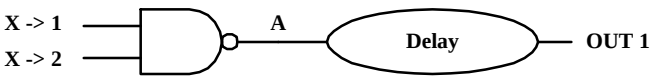
The 5 states simulated for gate outputs are logical 1, logical 0, 3-State, undefined, and unknown (see Figure 91). All logic elements understood by the simulator are positive logic .

State	Symbol	Meaning
Low	**0**	Logic state is low (false).
High	**1**	Logic state is high (true).
Unknown	*****	Logic state is not known, but is either low or high.
3-State	**3**	High impedance or floating state.
Undefined	**#**	Logic state is between low and high, indicating an unstable level

Figure 91

The simulation is based on two input files: a binary netlist (.BIN) file, and the Simulation Control Language (.SCL) file. The netlist specifies the logic and gate delay behavior of the circuit, while the .SCL file specifies the signal waveforms applied to the inputs of the circuit.

Figure 92 shows the simulation model for one type of element often used in netlists, the NAND gate. When the simulation starts, the stimuli description (.SCL file) is read, checked for syntax errors and converted to an intermediate format which is stored on the disk. The network is then loaded into the memory and simulation starts. During simulation, the status of all signals specified in the stimuli file print list is written to the result (.RES) file. This result file is the basis for the waveform viewer display.



NAND	0	1	*	3	#
0	**1**	**1**	**1**	**1**	**1**
1	**1**	**0**	*****	#	#
*	**1**	*****	*****	#	#
3	**1**	#	#	#	#
#	**1**	#	#	#	#

Model is a truth table and a time delay pair.
delay = T_{LH}
or = T_{HL}

Figure 92

XPLA Designer Simulator

The XPLA Designer logic simulator waveform entry tool automatically converts the project netlist (.NET or MOD) into a binary format (.BIN file).

The simulation files produced by XPLA Designer are

.scl
.bin
.net
.res
.mod
.scr

The .BIN file is the binary netlist with timing information. This file specifies the behavior of the logic in the simulator. Since it is in binary format, it cannot be edited or read directly. The .SCL file contains the input waveforms that are used in the simulation; for most designs it should be edited/created using the Waveform editor.

Simulation Control Language

Inside the simulation engine (SimScl), control of the simulator is accomplished by means of an ASCII text file known as the .SCL file. As noted earlier, this file can be most easily generated using the waveform editor. However, for designs that have a large number of vectors, it can also be generated with as text editor so long as it uses the SCL format. The format consists of the initial state of the input signals, which signals will be displayed (input and output), the duration of the simulation session, etc. Complete, detailed information is provided here on the internal format of the SCL file for users who wish to edit the file directly.

Let's look at an example of an SCL file. Suppose the design to be simulated has four inputs (inl, in2, in3, in4) and two outputs (out1 , out2). If you want to display all inputs and outputs, all inputs will initially be at logical 0, and will later transition to logical 1 (see Figure 93). The first input will transition at 500 time units, the second at 1000, the third at 1200 and the fourth at 2000. Supposing that the circuit will stabilize within 500 more time units, we will simulate beyond the required 2000 to 2500 units.

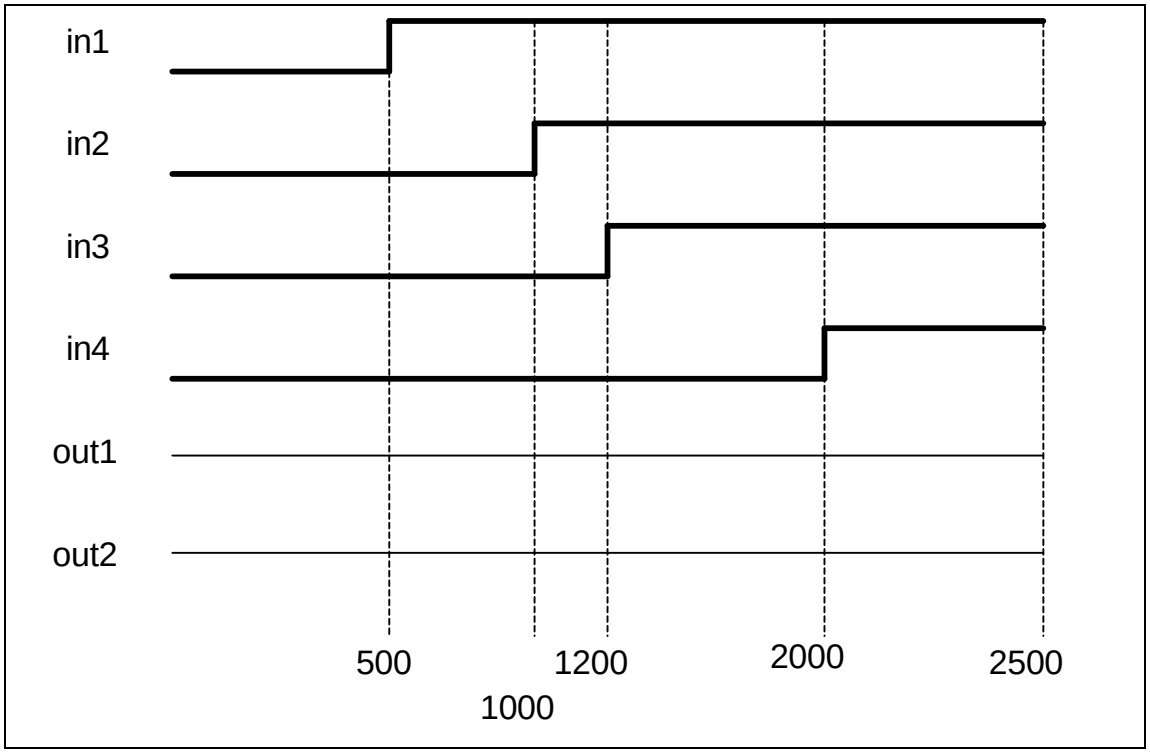


Figure 93

Figure 94 is an anatomy of SCL file which meets the basic format requirements and performs the task just outlined. The comment lines start with an asterisk. This example illustrates the "S", "SU" and "F" commands, which are further described later.

```
*  Simple SCL file (any comment is OK here)

S       0       (500)   in1                     -
*The above statement defines inl initially @ logic 0, and  transitions to 1 at 500 time units
* this will remain 1 for the remainder of the simulation

S       0       (1000) in2
*   similarly in2 transitions from 0 to 1 at 1000 times units

S       0       (1200) in3
*   in3 transitions from 0 to 1 at 1200

S       0       (2000) in4
* finally in4 transitions at 2000 —

SU time = 2500
*Defines the simulation time as 2500 total units

F
```

Figure 94

To use SCL files that were created using a text editor with the XPLA Designer simulator, you must first turn the off the 'autosave SCL file on run' feature inside the waveform editor. When the simulator waveform interface is running, this is an checkbox item under the options menu - it should be unchecked. Then you can load the SCL file created with a text editor and Run a simulation. The waveforms can be viewed on screen or printed.

Many good simulation practices could be summarized at the onset to help the designer develop good simulation practices, but one single cardinal rule will minimize frustration: **Always initialize flip-flops and registers!** This is important because any digital simulator will spend enormous amounts of time attempting to establish its initial state, which may be impossible.

Input Format

The input format of the stimuli file is as follows:

1. The scanning field is column 1 to 72 inclusive.
2. Lower case characters are treated as upper case.
3. Tabs are treated as blanks.
4. An asterisk (*) in column 1 means a comment line. A comment line has no influence on the program.

5. A number (#) in column 1 means a continuation line.
6. The statement field starts after at least one blank and consists of a keyword, usually followed by a further parameter, The keyword is separated from the parameter by at least one blank.
7. The label starts in column 1 and serves as a reference for a GT (go to) statement. It is separated from the statement by at least one blank. Labels within a subroutine are local.
8. Labels, signal names and variable names are composed of maximum 12 characters, the first being a letter.
9. <signal list> is a list of signal names, which are separated by commas.
10. <state list> is a compact string of logic states. No blanks are allowed in a state list, but commas may be used to separate the states.
11. Extra blanks are allowed anywhere except in names, numbers, and state lists.

General Information about Stimuli

SCL statements must be in the order shown below:

1. Subroutine declarations
2. Main part of stimuli file
3. Printlist definition
4. Finish command "F"

All commands must appear either in a subroutine or after the printlist definition (P command). An exception to this rule is the command IDENT or IDNT. It can be before the subroutine declaration.

Subroutines

1. Subroutines must be declared at the beginning of the stimuli file.
2. Subroutine declarations cannot be nested.
3. Before a subroutine can be called, it must be declared.
4. Subroutine names must be unique.
5. Subroutines can contain only local jumps. Labels declared inside a subroutine are unknown at the outside.
6. After the end of a subroutine declaration, all jumps are tested for correctness. Jumps that are not local (that is, where the label mentioned is unknown to the subroutine) will result in an error message.
7. Subroutine call nesting depth is limited to 20.
8. These commands belong to subroutines only : SUB, RET, END

Data Fields

1. A data field consists of data lines between the keywords DATS and DATE.
2. No other commands should be inserted in the data field.
3. More than one data field can be declared. The data lines are collected to form one big data field.
4. Each data field starts with the default data base BIN = binary.
5. The data values are stored as 32 Bit values.
6. These commands belong to data fields: DATE, DATS, DATV, SDC, INCR.

Variables

1. After using the finish command F, all variables are checked to see that they have been assigned a value. If not, an error message is written in <design>.err together with their names.
2. These commands assign a value to a variable: DATV, SETV.
3. Variables are 32 Bits long.
4. The following commands can be used with variables: DATV, DECV, IF, IFV, INCR, PV, SETV

Jumps and Labels

1. Labels are local to a subroutine. They are unknown outside subroutines.
2. Jumps into a subroutine are not permitted.
3. Jumps out of a subroutine are not permitted.
4. At the end of a subroutine declaration, all jumps are checked. Nonlocal jumps create an error message.
5. After the finish command F, all jumps in the main part are checked. Nonexistent labels create error messages.
6. Forward and backward jumps are possible.
7. There is one command related to jumps: GT

SCL Keyword Summary

Keyword	Explanation
BUSI	switch the peripheral bus line to input state
BUSO	switch the peripheral bus line to output state
CALL	call a subroutine
DATE	end data lines
DATS	start data lines
DATV	assign a data line to a variable
DECV	decode a variable
END	end a subroutine declaration
F	finish
GT	go to
IF	if statement
RET	return from a subroutine
S	sequence
SDC	set data counter
SETV	set a variable
ST	set to
STAB	stability check
ST DATA	set to data
SU	simulate until
SUB	start of a subroutine declaration
SUNS	simulate until network stable
TRAC	transition check

SCL Keyword Definitions

BUSI Switch Bus Line to Input State

Format

BUSI <signal list>

Only NETIO signals (output of bus function) are allowed.

The output of a bus line with a name that occurs in the signal list is changed to the input state condition, i.e., a forcing external stimulus may be applied to that node. In this situation the simulator checks whether the inputs of the bus line are kept on real or virtual

3-State (pull-up or pull-down) resistor. If this is not the case, the simulator produces an undefined state.

BUSO **Switch Bus Line to Output**

Format

BUSO <signal list>

Only NETIO signals (output of bus function) are allowed.

The output of a bus line with a name that occurs in the signal list is changed to the output state condition. In this situation the simulator checks whether a forcing external stimulus is applied to that node. If this is the case, the simulator produces an undefined state.

Note: At the beginning of simulation all bus lines are treated as ordinary signals, and can be forced to any state designed. A check is made after the first appearance of either BUSI for BUSO. It is good practice to give a BUSO command for all NETIO signals at the very beginning of the simulation.

CALL **Subroutine Call**

Format

CALL <subname>

This is used to execute a number of SCL statements specified by the subroutine <subname>. Subroutines must be declared before their call.

DATE **End of data lines**

Data lines must appear between these two keywords and must not be interrupted by other commands. Data line information starts after column 1. If several data strings occur in the same line, they must be separated by a slash (/). The data must be given in the following data bases:

Keyword	
BIN	**binary**
OCT	**octal**
HEX	**Hexidecimal**

It is possible to switch between these data bases. If no data base is given, BIN is assumed as the default data base.

It is possible to have several data fields. Every new field (enclosed between DATS/ DATE commands) is added to the existing ones.

If the number of values of a data string is fewer than the number of signals in the ST DATA command, the last-mentioned value is taken for the remaining signals.

Example:

```
DATS
        011/OCT/765/11/HEX/1A
        7C4/BIN/11101/10
DATE
  |
  |
  |
DATS
        10101/HEX/1A/1C/1F
        OCT/73/26/BIN/0010
DATE
```

DATS **start of data lines**

DATV **assign Data Lines to a Variable**

Format

DATV <variable list>

<variable list>: list of variable names, separated by one comma.

The data of the data field is converted to integers and assigned to the corresponding variables. The data entry indicated by the data counter is assigned to the first variable of the variable list. Then the data counter is incremented by one.

This process is repeated until all variables of the list have been assigned a value. If there are fewer data entries in the data field than variables in the list, the last data value is assigned to the remaining variables.

Example

```
DATS
      BIN/10/111                  *Data field with 2 entries
DATE
SDC1                              *Set data counter to 1 (default)

DATV VAR1, VAR2, VAR3, VAR4       *assign values to variables
                                  VAR1 = 2
                                  VAR2 = 7
                                  VAR3 = 7
                                  VAR4 = 7
```

DECV Decode a Variable

Format

DECV <variable> (<signal list>)

<variable list> : list of variable names, separated by one comma
<signal list> : list of signal names, separated by one comma.

The content of the variable is decoded when the statement is executed. The non-negative value of the variable is converted into a binary number. These bits are assigned to the signal list. The least significant bit is assigned to the rightmost signal.

If necessary, leading zeros are inserted.

This statement influences the simulation in the same way as the "ST" command.

Example 1

```
SETV VAR1,6              *assign value "6" to VAR1
                         *(binary value "110")

DECV VAR1 (A, B, C, D)   *will result in :

                         *A = 0  (leading zero inserted)
                         *B = 1
                         *C = 1
                         *D = 0 (least significant bit)
```

Example 2

```
SET VAR2,11              *assign value "11" to VAR2
                         *(binary value "1011)

DEC VAR2 (A, B)          *will result in:
                         *A =   1
      * (right most signal)  B =   1 (least significant bit)
```

END — End of a Subroutine Declaration

Format

END

END is the last statement of a subroutine. The program then goes to the next statement. END must not be omitted.

```
Sub test 1          Sub test 1
    |                   |
    |                   |
    |                   |
  Ret                   |
  End                 End
```

Both of these examples show valid subroutine declarations.

F **Finish**

Format

F

The Finish command terminates the simulation and must be the last command of a stimuli file. Entries after the F command are not processed.

GT **Goto Statement**

Format

GT <label>

<label> is a string of up to 12 alphanumeric characters, the first being a letter.

Note: Labels start in column 1. The GT statement causes a jump to the statement identified by <label>. Labels defined within a subroutine are local. Jumps out of or into a subroutine are not permitted.

At the end of a subroutine declaration the labels are checked again. Therefore, illegal jumps are detected only after the subroutine has been closed.

The same holds for the main part of the stimuli file and the Finish keyword. After reading the F keyword, labels and jumps of the main part are checked and errors announced.

Labels must be unique, with one exception: the same label name can appear in a subroutine and in the main part of the stimuli file.

Example

```
        |
        |
LABEL1 ST 101 (A, B, C)
SUNS
        |
        |
GT LABEL1
or
LABEL2
ST 101 (A, B, C)
SUNS
        |
        |
GT LABEL2
```

IF **IF Statement**

Format

IF TIME <relation> <ntime> <executable SCL statement>

<relation>: " < " or " = " or " > "
<ntime> : absolute time slot, can be a constant or a variable

If the condition of the IF-clause is true when read, the SCL-statement is executed.

Note: Nesting of IF-statements is not allowable.

Example

```
IF TIME = 10000 CALL SUBTEST 1
* if the actual  timeslot equals 100000, the subroutine SUBTEST1 is called

IF TIME = VAR1 GT RESET1
*If the actual timeslot equals the value of variable VAR1, then the program jumps to label
RESET1
```

IFV Condition Check on a Variable

Format

IFV <variable> <relation> [<sign>] <value> <executable SCL-statement>

<variable> : variable name of up to 12 characters, the first being a letter.
<relation> : "<" or "=" or **">"**
<sign> : minus sign (optional).
-- if no sign is given, a positive value is assumed.
<value> : a positive constant or variable
If the condition of the I F-clause is true when read, the SCL-statement is executed.

Note: Nesting of IFV-statements is not allowed.

Example

```
IFV VAR1 = 10 CALL SUBTEST1
*If the value of variable VAR1 equals 10, the subroutine SUBTEST1 is called.

IFV VAR2  < VAR3 GT RESET1>
*If the value of variable VAR2 is greater than the value of variable VAR3, the program
jumps to label RESET1.
```

INCR Increment a Variable with a Specified Value

Format

INCR <variable>, [<sign>] value

<variable> : variable name up to 12 characters, the first being a letter.
<sign> : minus sign (optional). If no sign is given, a positive value is assumed.
<value> : a positive constant or variable.

Example 1

```
INCR  COUNT,10        -- value of variable COUNT is incremented by 10
INCR  COUNT,-3        -- value of variable COUNT is decremented by 3
```

Example 2

SETV VAR1, 3	-- variable VAR1 set to 3
SETV COUNT, 10	-- variable COUNT set to 10
INCR COUNT, VAR1	--value of variable COUNT incremented by value of variable VAR1: COUNT = 13.

IT **Initialize To**

Format

IT <initstate list> (<signal list>)

Initializes all signals of <signal list> to the states given in <initstate list>. All other signals are set to unknown.

The k-th state corresponds with the k-th signal. When not enough states have been specified, the last state is repeated. The simulator clock is set on time slot -0.

Note: The IT command must appear only once, at the beginning of the program. Otherwise it is treated as ST.

Example

IT 01 (Al B, C, D)	--result: A=0, B=C=D=l, all others at unknown (*)

Note: All NETIN signals (primary inputs) should be initialized at the beginning.

LIST **Listing of Signals with State "*" "#"**

Format

LIST <specification>

<specification> : UNDEF OR NOTDET

Writes a list of all signals that are currently not determined ("*") or not defined ("#"). When the LIST statement in a stimuli file is read, all signals which have the specified state are listed in the file <design>.err.

"List is blank" means that no signals meet the condition.

Example:

```
        |
        |
SU TIME = 10000
LIST UNDEF
LIST NOTDET
```

Contents of <design>.err:

```
        |
        |
LIST UNDEF . . . .Undefined signals at timeslot 100000:
signall signal2 signal3 signal4 signal5 signal6


LIST NODET . . . . Unknown signals at timeslot 10000:
List is blank
```

RET **Return From Subroutine**

Format

RET:

This is only used within a subroutine. When executed, control is passed to the next statement after the subroutine call.

S Sequence

Format

S initstate (timel, time2.....) <signal list>
S initstate (timel , time2,....., ETC) <signal list>

The sequence command causes a series of signal changes dependent on the simulator clock.

initstate = 0 or 1
0 < timei < timei + 1 = time relative to current simulator time.

When the sequence formed by the relative time slots arrives at ETC, the sequence is repeated.

A sequence is canceled if IT, ST, or S is specified for that particular signal.

Example

```
SO      (5, 10,  ETC) A, B
SO      (5, 10, 15, ETC) C
SO      (5, 10, 15f 25, ETC) D
Sl      (5) E
Sl      (5, 10) F
```

These commands were put in at time = 100. The output is:

Time	A	B	C	D	E	F
100	0	0	0	0	1	1
105	1	1	1	1	0	0
110	0	0	0	0	0	1
115	1	1	0	1	0	1
120	0	0	1	1	0	1
125	1	1	0	0	0	1
130	0	0	0	1	0	1

The sequences for signals form A to D are repeated. The other sequences are canceled after the last change.

Note: During simulation the number of active sequences has a maximum limit, depending on the size of the sequences.

SDC Set Data Counter

Format

SDC <value>

<value> may be a constant or a variable. Variables can be manipulated via the SETV or the INCR command.

This command is used to reinitialize the data counter. The simulator starts with the value 1 for the data counter. When the counter exceeds the number of available data lines, the simulation is terminated.

Example

```
SDC 10    --  the next ST DATA command gets its data from data string number 10
SDC count1   -- the data counter gets the value of count1 at the moment of execution
```

SETV **Set a Variable to a Value**

Format

SETV <variable> [<sign>] <value>

<variable> : variable name of up to 12 characters, the first being a letter.
<sign> : minus sign (optional). If no sign is given, a positive value is assumed.
<value> : a positive constant or variable.

Examples

```
SETV VARI, 10        -- variable VAR1 set ro value 10
SETV VAR1, -3        -- variable VAR1 set to value -3
SETV VAR1, VAR2  -- variable VAR1 set to value of variable VAR2
SETV VAR1, -VAR2 -- variable VAR1 set to (negative) value of variable VAR2
```

ST **Set To**

Format

ST <state list> (<signal list>)

This has the same meaning as the IT command, except that the non-specified signals and the simulator clock remain unchanged.

Example

```
ST 13** (Al B, C, D)          -- result: A = 1, B = 3,  C = D = * ; all others are unchanged
```

STAB **Stability Check**

Format

STAB <n>

During simulation the stability of the network is checked at the moment signals are forced by an IT, ST, ST DATA, DECV or S command. The messages are printed in alphanumeric mode.

<n> is an optional parameter (default is 25) giving the number of messages. If <n> is exceeded, the STAB check is canceled. STAB may appear anywhere in the program.

ST DATA **Set to DATA**

Format

ST DATA (<signal list>)

This command has the same function as the ST command. The string of states, identified by the current data counter, is taken from the data field(s). These are strings of data specified between the keywords DATS and DATE. The data counter is the data string number.

Each ST DATA command causes the data counter to be incremented by 1. If the string of states is not sufficient for the signal list, the last state is repeated for the remainder of the signals.

Example

* Data field given as :

```
DATS
101/HEX/1A/FC
BIN/111/10
DATE
Data counter set to 4:
ST DATA (A, B, C, D)
*                    will result in          A set to 1
```

```
                                        B set to 1
                                        C set to 0
                                        D set to 1

* Now the data counter is set to 5.

* Next
ST DATA (A, B, C, D)
*                   will result in      A set to 1
                                        B set to 1
                                        C set to 0
                                        D set to 1

* Next the data counter is automatically set to 5.
ST DATA (A, B, C, D)
*                   will result in      A set to 1
                                        B set to 1
                                        C set to 0
                                        D set to 1

*Next the data counter is automatically set to 5.

ST DATA (A, B, C, D)
*                   will result in      A set to 1
                                        B set to 0
                                        C set to 0  - Last State is repeated
                                        D set to 0  -     "
```

SU **Simulate Until**

SUB **Start of a Subroutine Declaration**

Format

SUB <subname>

This is the first statement of a subroutine. The <subname> is a string of at most 12 alphanumeric characters starting with a letter. It identifies the subroutine and must be unique.

The subroutine facility enables the user to execute some SCL statements several times in difference parts of the file by calling them by the subroutine-name.

A subroutine declaration is not allowed within a subroutine, but one subroutine may call another.

Notes-

1. The simulator requires that all subroutines be declared at the beginning of the program.
2. If subroutine SUB1 calls subroutine SUB2, then subroutine SUB2 must be declared somewhere above subroutine SUB1.

SUNS Simulate Until the Network is Stable

There are three possible formats

1. SU TIME = stop time
   Stop time = absolute time slot, can be constant or a variable.

2. SU TIME = *+delta time
   Delta time = number of time slots after the current time slot.

3. SUNS

In case 1 above, simulation starts at the actual time slot and proceeds to the absolute time slot.

In case 2 above**,** simulation starts at the actual time slot and process to actual + delta time slot.

In case 3 above, simulation starts at the actual time slot and continues until there are no more internal changes in the network.

Example

```
SU TIME = 1600
```

Assuming an actual time slot of 1000, then simulation continues to time slot 1600**.**

Example

SU TIME = *+500

Assuming an actual time slot of 1 000, then simulation continues to time slot 1000 + 500 = 1500.

Note: No blanks are allowed between *+ and delta time.

TRAC Transition Check

Format

Trac

During simulation, signals are checked to see if they change from O to 1 and/or from 1 to 0. The result is printed on special demand.

The netlist is checked throughout the simulation, regardless of the position of TRAC in the stimuli file.

Generating Clocks

Let's assume you wish to make two clocks CLK1 and a second clock (CLK2) which is at 1/2 the frequency of CLK1. The period of CLK1 is 30 ns and CLK2 is 60 ns. We will define CLK2 as one for 15 ns and zero for 15 ns, and CLK2 as zero for 30 ns and one for 30 ns. Also, 1 unit = 1 ns for this example. Figure 95 shows the timing, and Figure 96 shows the corresponding SCL file.

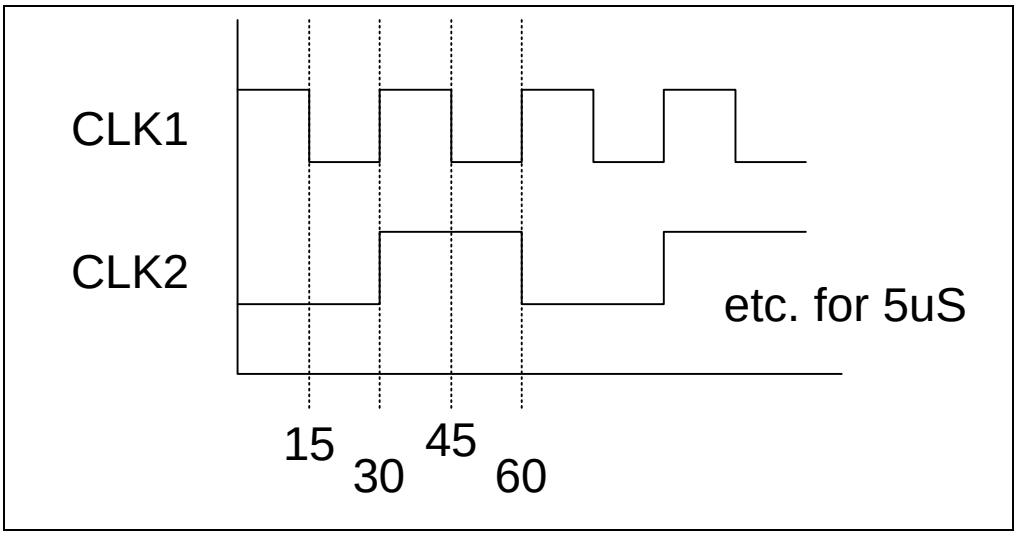


Figure 95

```
**
S 1 (15, 30, ETC)          *CLK1
S 0 (30, 60, ETC)          *CLK2
SU time = 5000
F
```

Figure 96

Subroutine Example

As an example of subroutine usage, the following SCL file generates three bursts of input in three consecutive 1000-nanosecond time periods.

```
SUB  STIMULI1
S       0       (50, 100, 150, 200) IN1
S       0       (50, 200) IN2
S       0       (100, 200, 300, 400) IN3
S       1       (50) IN4
RET
END

IT 0 (IN1, IN2, IN3, IN4)
SU TIME=*+1000
CALL STIMULI1
SU TIME=*+1000
CALL STIMULI1
SU TIMR=*+1000
CALL STIMULI1
SU TIME=*+1000
*
F
```

Appendix B *Device Pin Out Configurations*

The following pages list pin and node numbers for the various package and macrocell count combinations of Philips CPLDs supported by the XPLA Device Kit.

PZ3032/PZ5032 44 Pin PLCC

Pin	Name	Node	Pin	Name	Node
1	**IN1**		**23**	**V_{DD}**	
2	**IN3**		**24**	**B15**	**76**
3	**V_{DD}**		**25**	**B14**	**75**
4	**A0/CLK1**	**45**	**26**	**B13**	**74**
5	**A1**	**46**	**27**	**B12**	**73**
6	**A2**	**47**	**28**	**B11**	**72**
7	**A3**	**48**	**29**	**B10**	**71**
8	**A4**	**49**	**30**	**GND**	
9	**A5**	**50**	**31**	**B9**	**70**
10	**GND**		**32**	**B8**	**69**
11	**A6**	**51**	**33**	**B7**	**68**
12	**A7**	**52**	**34**	**B6**	**67**
13	**A8**	**53**	**35**	**V_{DD}**	
14	**A9**	**54**	**36**	**B5**	**66**
15	**V_{DD}**		**37**	**B4**	**65**
16	**A10**	**55**	**38**	**B3**	**64**
17	**A11**	**56**	**39**	**B2**	**63**
18	**A12**	**57**	**40**	**B1**	**62**
19	**A13**	**58**	**41**	**B0**	**61**
20	**A14**	**59**	**42**	**GND**	
21	**A15**	**60**	**43**	**IN0/CLK0**	
22	**GND**		**44**	**IN2**	

PZ3032/PZ5032 44 Pin TQFP

Pin	Name	Node	Pin	Name	Node
1	**A3**	**48**	**23**	**B10**	**71**
2	**A4**	**49**	**24**	**GND**	
3	**A5**	**50**	**25**	**B9**	**70**
4	**GND**		**26**	**B8**	**69**
5	**A6**	**51**	**27**	**B7**	**68**
6	**A7**	**52**	**28**	**B6**	**67**
7	**A8**	**53**	**29**	**V_{DD}**	
8	**A9**	**54**	**30**	**B5**	**66**
9	**V_{DD}**		**31**	**B4**	**65**
10	**A10**	**55**	**32**	**B3**	**64**
11	**A11**	**56**	**33**	**B2**	**63**
12	**A12**	**57**	**34**	**B1**	**62**
13	**A13**	**58**	**35**	**B0**	**61**
14	**A14**	**59**	**36**	**GND**	
15	**A15**	**60**	**37**	**IN0/CLK0**	
16	**GND**		**38**	**IN2**	
17	**V_{DD}**		**39**	**IN1**	
18	**B15**	**76**	**40**	**IN3**	
19	**B14**	**75**	**41**	**V_{DD}**	
20	**B13**	**74**	**42**	**A0/CLK1**	**45**
21	**B12**	**73**	**43**	**A1**	**46**
22	**B11**	**72**	**44**	**A2**	**47**

PZ3064/PZ5064 44 Pin PLCC

Pin	Name	Node	Pin	Name	Node
1	**IN1**		**23**	**V_{DD}**	
2	**IN3**		**24**	**C0/CLK1**	**77**
3	**V_{DD}**			**C1**	**78**
4	**A0/CLK3**	**45**	**25**	**C2**	**79**
	A1	**46**	**26**	**C3**	**80**
5	**A2**	**47**	**27**	**C4**	**81**
	A3	**48**		**C5**	**82**
	A4	**49**		**C6**	**83**
6	**A5**	**50**	**28**	**C7**	**84**
	A6	**51**	**29**	**C8**	**85**
	A7	**52**		**C9**	**86**
7	**A8**	**53**		**C10**	**87**
	A9	**54**		**C11**	**88**
	A10	**55**		**C12**	**89**
8	**A11**	**56**	**30**	**GND**	
9	**A12**	**57**	**31**	**C13**	**90**
10	**GND**			**C14**	**91**
11	**A13**	**58**	**32**	**C15**	**92**
	A14	**59**	**33**	**D15**	**108**
12	**A15**	**60**		**D14**	**107**
13	**B15**	**76**	**34**	**D13**	**106**
	B14	**75**	**35**	**V_{DD}**	
14	**B13**	**74**	**36**	**D12**	**105**
15	**V_{DD}**		**37**	**D11**	**104**
	B12	**73**		**D10**	**103**
	B11	**72**		**D9**	**102**
16	**B10**	**71**	**38**	**D8**	**101**
	B9	**70**	**39**	**D7**	**100**
17	**B8**	**69**		**D6**	**99**
	B7	**68**		**D5**	**98**
	B6	**67**		**D4**	**97**
	B5	**66**		**D3**	**96**
18	**B4**	**65**	**40**	**D2**	**95**
19	**B3**	**64**		**D1**	**94**
20	**B2**	**63**	**41**	**D0**	**93**
	B1	**62**	**42**	**GND**	
21	**B0/CLK2**	**61**	**43**	**IN0/CLK0**	
22	**GND**		**44**	**IN2**	

PZ3064/PZ5064 44 Pin TQFP

Pin	Name	Node	Pin	Name	Node
	A6	**71**	**23**	**C8**	**105**
	A7	**72**		**C9**	**106**
1	**A8**	**73**		**C10**	**107**
	A9	**74**		**C11**	**108**
	A10	**75**		**C12**	**109**
2	**A11**	**76**	**24**	**GND**	
3	**A12**	**77**	**25**	**C13**	**110**
4	**GND**			**C14**	**111**
5	**A13**	**78**	**26**	**C15**	**112**
	A14	**79**	**27**	**D15**	**128**
6	**A15**	**80**		**D14**	**127**
7	**B15**	**96**	**28**	**D13**	**126**
	B14	**95**	**29**	**V_{DD}**	
8	**B13**	**94**	**30**	**D12**	**125**
9	**V_{DD}**		**31**	**D11**	**124**
	B12	**93**		**D10**	**123**
	B11	**92**		**D9**	**122**
10	**B10**	**91**	**32**	**D8**	**121**
	B9	**90**	**33**	**D7**	**120**
11	**B8**	**89**		**D6**	**119**
	B7	**88**		**D5**	**118**
	B6	**87**		**D4**	**117**
	B5	**86**		**D3**	**116**
12	**B4**	**85**	**34**	**D2**	**115**
13	**B3**	**84**		**D1**	**114**
14	**B2**	**83**	**35**	**D0**	**113**
	B1	**82**	**36**	**GND**	
15	**B0/CLK2**	**81**	**37**	**IN0/CLK0**	
16	**GND**		**38**	**IN2**	
17	**V_{DD}**		**39**	**IN1**	
18	**C0/CLK1**	**97**	**40**	**IN3**	
	C1	**98**	**41**	**V_{DD}**	
19	**C2**	**99**	**42**	**A0/CLK3**	**65**
20	**C3**	**100**		**A1**	**66**
21	**C4**	**101**	**43**	**A2**	**67**
	C5	**102**		**A3**	**68**
	C6	**103**		**A4**	**69**
22	**C7**	**104**	**44**	**A5**	**70**

PZ3064/PZ5064 68 Pin PLCC

Pin	Name	Node	Pin	Name	Node
1	**IN1**		**35**	**V_{DD}**	
2	**IN3**		**36**	**C0/CLK1**	**101**
3	**V_{DD}**			**C1**	**102**
4	**A0/CLK3**	**69**	**37**	**C2**	**103**
	A1	**70**	**38**	**GND**	
5	**A2**	**71**	**39**	**C3**	**104**
6	**GND**		**40**	**C4**	**105**
7	**A3**	**72**	**41**	**C5**	**106**
8	**A4**	**73**		**C6**	**107**
9	**A5**	**74**	**42**	**C7**	**108**
	A6	**75**	**43**	**V_{DD}**	
10	**A7**	**76**	**44**	**C8**	**109**
11	**V_{DD}**			**C9**	**110**
12	**A8**	**77**	**45**	**C10**	**111**
	A9	**78**	**46**	**C11**	**112**
13	**A10**	**79**	**47**	**C12**	**113**
14	**A11**	**80**	**48**	**GND**	
15	**A12**	**81**	**49**	**C13**	**114**
16	**GND**			**C14**	**115**
17	**A13**	**82**	**50**	**C15**	**116**
	A14	**83**	**51**	**D15**	**132**
18	**A15**	**84**		**D14**	**131**
19	**B15**	**100**	**52**	**D13**	**130**
	B14	**99**	**53**	**V_{DD}**	
20	**B13**	**98**	**54**	**D12**	**129**
21	**V_{DD}**		**55**	**D11**	**128**
22	**B12**	**97**		**D10**	**127**
23	**B11**	**96**	**56**	**D9**	**126**
24	**B10**	**95**	**57**	**D8**	**125**
	B9	**94**	**58**	**GND**	
25	**B8**	**93**	**59**	**D7**	**124**
26	**GND**		**60**	**D6**	**123**
27	**B7**	**92**		**D5**	**122**
	B6	**91**	**61**	**D4**	**121**
28	**B5**	**90**	**62**	**D3**	**120**
29	**B4**	**89**	**63**	**V_{DD}**	
30	**B3**	**88**	**64**	**D2**	**119**
31	**V_{DD}**			**D1**	**118**
32	**B2**	**87**	**65**	**D0**	**117**
	B1	**86**	**66**	**GND**	
33	**B0/CLK2**	**85**	**67**	**IN0/CLK0**	
34	**GND**		**68**	**IN2**	

PZ3064/PZ5064 84 Pin PLCC

PIN	NAME	NODE	PIN	NAME	NODE	PIN	NAME	NODE
1	**IN1**		**29**	**B10**	**111**	**57**	**C11**	**128**
2	**IN3**		**30**	**B9**	**110**	**58**	**C12**	**129**
3	**V_{DD}**		**31**	**B8**	**109**	**59**	**GND**	
4	**A0/CLK3**	**85**	**32**	**GND**		**60**	**C13**	**130**
5	**A1**	**86**	**33**	**B7**	**108**	**61**	**C14**	**131**
6	**A2**	**87**	**34**	**B6**	**107**	**62**	**C15**	**132**
7	**GND**		**35**	**B5**	**106**	**63**	**D15**	**148**
8	**A3**	**88**	**36**	**B4**	**105**	**64**	**D14**	**147**
9	**A4**	**89**	**37**	**B3**	**104**	**65**	**D13**	**146**
10	**A5**	**90**	**38**	**V_{DD}**		**66**	**V_{DD}**	
11	**A6**	**91**	**39**	**B2**	**103**	**67**	**D12**	**145**
12	**A7**	**92**	**40**	**B1**	**102**	**68**	**D11**	**144**
13	**V_{DD}**		**41**	**B0/CLK2**	**101**	**69**	**D10**	**143**
14	**A8**	**93**	**42**	**GND**		**70**	**D9**	**142**
15	**A9**	**94**	**43**	**V_{DD}**		**71**	**D8**	**141**
16	**A10**	**95**	**44**	**C0/CLK1**	**117**	**72**	**GND**	
17	**A11**	**96**	**45**	**C1**	**118**	**73**	**D7**	**140**
18	**A12**	**97**	**46**	**C2**	**119**	**74**	**D6**	**139**
19	**GND**		**47**	**GND**		**75**	**D5**	**138**
20	**A13**	**98**	**48**	**C3**	**120**	**76**	**D4**	**137**
21	**A14**	**99**	**49**	**C4**	**121**	**77**	**D3**	**136**
22	**A15**	**100**	**50**	**C5**	**122**	**78**	**V_{DD}**	
23	**B15**	**116**	**51**	**C6**	**123**	**79**	**D2**	**135**
24	**B14**	**115**	**52**	**C7**	**124**	**80**	**D1**	**134**
25	**B13**	**114**	**53**	**V_{DD}**		**81**	**D0**	**133**
26	**V_{DD}**		**54**	**C8**	**125**	**82**	**GND**	
27	**B12**	**113**	**55**	**C9**	**126**	**83**	**IN0/CLK0**	
28	**B11**	**112**	**56**	**C10**	**127**	**84**	**IN2**	

PZ3064/PZ5064 100 Pin PQFP

PIN	NAME	NODE	PIN	NAME	NODE	PIN	NAME	NODE
1	**NC**		**35**	**B3**	**120**	**69**	**D12**	**161**
2	**NC**		**36**	**V_{DD}**		**70**	**D11**	**160**
3	**A6**	**107**	**37**	**B2**	**119**	**71**	**D10**	**159**
4	**A7**	**108**	**38**	**B1**	**118**	**72**	**NC**	
5	**V_{DD}**		**39**	**B0/CLK2**	**117**	**73**	**D9**	**158**
6	**A8**	**109**	**40**	**GND**		**74**	**NC**	
7	**NC**		**41**	**V_{DD}**		**75**	**D8**	**157**
8	**A9**	**110**	**42**	**C0/CLK1**	**133**	**76**	**GND**	
9	**NC**		**43**	**C1**	**134**	**77**	**D7**	**156**
10	**A10**	**111**	**44**	**C2**	**135**	**78**	**D6**	**155**
11	**A11**	**112**	**45**	**GND**		**79**	**NC**	
12	**A12**	**113**	**46**	**C3**	**136**	**80**	**NC**	
13	**GND**		**47**	**C4**	**137**	**81**	**D5**	**154**
14	**A13**	**114**	**48**	**C5**	**138**	**82**	**D4**	**153**
15	**A14**	**115**	**49**	**C6**	**139**	**83**	**D3**	**152**
16	**A15**	**116**	**50**	**C7**	**140**	**84**	**V_{DD}**	
17	**B15**	**132**	**51**	**NC**		**85**	**D2**	**151**
18	**B14**	**131**	**52**	**NC**		**86**	**D1**	**150**
19	**B13**	**130**	**53**	**V_{DD}**		**87**	**D0**	**149**
20	**V_{DD}**		**54**	**C8**	**141**	**88**	**GND**	
21	**B12**	**129**	**55**	**NC**		**89**	**IN0/CLK0**	
22	**B11**	**128**	**56**	**C9**	**142**	**90**	**IN2**	
23	**B10**	**127**	**57**	**NC**		**91**	**IN1**	
24	**NC**		**58**	**C10**	**143**	**92**	**IN3**	
25	**B9**	**126**	**59**	**C11**	**144**	**93**	**V_{DD}**	
26	**NC**		**60**	**C12**	**145**	**94**	**A0/CLK3**	**101**
27	**B8**	**125**	**61**	**GND**		**95**	**A1**	**102**
28	**GND**		**62**	**C13**	**146**	**96**	**A2**	**103**
29	**NC**		**63**	**C14**	**147**	**97**	**GND**	
30	**NC**		**64**	**C15**	**148**	**98**	**A3**	**104**
31	**B7**	**124**	**65**	**D15**	**164**	**99**	**A4**	**105**
32	**B6**	**123**	**66**	**D14**	**163**	**100**	**A5**	**106**
33	**B5**	**122**	**67**	**D13**	**162**			
34	**B4**	**121**	**68**	**V_{DD}**				

PZ3128/PZ5128 84 Pin PLCC

PIN	NAME	NODE	PIN	NAME	NODE	PIN	NAME	NODE
1	**IN1**		**30**	**C4**	**121**	**57**	**F7**	**172**
2	**IN3**			**C3**	**120**		**F8**	**173**
3	**V_{DD}**		**31**	**C2**	**119**		**F9**	**174**
4	**A15/CLK3**	**100**		**C1**	**118**	**58**	**F10**	**175**
	A14	**99**		**C0**	**117**	**59**	**GND**	
5	**A13**	**98**	**32**	**GND**			**F11**	**176**
6	**A12**	**97**	**33**	**D15**	**148**	**60**	**F12**	**177**
7	**GND**			**D14**	**147**	**61**	**F13**	**178**
	A11	**96**		**D13**	**146**		**F14**	**179**
8	**A10**	**95**	**34**	**D12**	**145**	**62**	**F15 / TCK**	**180**
	A9	**94**		**D11**	**144**	**63**	**G0**	**181**
	A8	**93**	**35**	**D10**	**143**		**G1**	**182**
9	**A7**	**92**		**D9**	**142**	**64**	**G2**	**183**
	A6	**91**	**36**	**D8**	**141**		**G3**	**184**
10	**A5**	**90**	**37**	**D7**	**140**	**65**	**G4**	**185**
11	**A4**	**89**		**D6**	**139**	**66**	**V_{DD}**	
	A3	**88**		**D5**	**138**		**G5**	**186**
12	**A2**	**87**	**38**	**V_{DD}**			**G6**	**187**
	A1	**86**	**39**	**D4**	**137**	**67**	**G7**	**188**
	A0	**85**		**D3**	**136**	**68**	**G8**	**189**
13	**V_{DD}**		**40**	**D2**	**135**		**G9**	**190**
14	**B15 / TDO**	**116**		**D1**	**134**	**69**	**G10**	**191**
	B14	**115**	**41**	**D0/CLK2**	**133**		**G11**	**192**
	B13	**114**	**42**	**GND**		**70**	**G12**	**193**
15	**B12**	**113**	**43**	**V_{DD}**			**G13**	**194**
	B11	**112**	**44**	**E0/CLK1**	**149**		**G14**	**195**
16	**B10**	**111**		**E1**	**150**	**71**	**G15 / TDO**	**196**
	B9	**110**	**45**	**E2**	**151**	**72**	**GND**	
17	**B8**	**109**		**E3**	**152**		**H0**	**197**
18	**B7**	**108**	**46**	**E4**	**153**		**H1**	**198**
19	**GND**		**47**	**GND**		**73**	**H2**	**199**
	B6	**107**		**E5**	**154**		**H3**	**200**
	B5	**106**		**E6**	**155**	**74**	**H4**	**201**
20	**B4**	**105**	**48**	**E7**	**156**	**75**	**H5**	**202**
	B3	**104**	**49**	**E8**	**157**		**H6**	**203**
21	**B2**	**103**		**E9**	**158**	**76**	**H7**	**204**
	B1	**102**	**50**	**E10**	**159**		**H8**	**205**
22	**B0**	**101**		**E11**	**160**		**H9**	**206**
23	**C15 / TMS**	**132**	**51**	**E12**	**161**	**77**	**H10**	**207**
	C14	**131**		**E13**	**162**	**78**	**V_{DD}**	
24	**C13**	**130**		**E14**	**163**		**H11**	**208**
25	**C12**	**129**	**52**	**E15**	**164**	**79**	**H12**	**209**
26	**V_{DD}**		**53**	**V_{DD}**		**80**	**H13**	**210**
	C11	**128**		**F0**	**165**		**H14**	**211**
27	**C10**	**127**		**F1**	**166**	**81**	**H15**	**212**
	C9	**126**	**54**	**F2**	**167**	**82**	**GND**	
	C8	**125**		**F3**	**168**	**83**	**IN0/CLK0**	
28	**C7**	**124**	**55**	**F4**	**169**	**84**	**IN2**	
	C6	**123**	**56**	**F5**	**170**			
29	**C5**	**122**		**F6**	**171**			

PZ3128/PZ5128 100 Pin PQFP

PIN	NAME	NODE	PIN	NAME	NODE	PIN	NAME	NODE
1	**A5**	**106**	**34**	**D7**	**156**	**67**	**G4**	**201**
2	**A4**	**105**		**D6**	**155**	**68**	**V_{DD}**	
	A3	**104**	**35**	**D5**	**154**	**69**	**G5**	**202**
3	**A2**	**103**	**36**	**V_{DD}**			**G6**	**203**
	A1	**102**	**37**	**D4**	**153**	**70**	**G7**	**204**
4	**A0**	**101**		**D3**	**152**	**71**	**G8**	**205**
5	**V_{DD}**		**38**	**D2**	**151**		**G9**	**206**
6	**B15 / TDI**	**132**		**D1**	**150**	**72**	**G10**	**207**
	B14	**131**	**39**	**D0-CLK2**	**149**		**G11**	**208**
7	**B13**	**130**	**40**	**GND**		**73**	**G12**	**209**
8	**B12**	**129**	**41**	**V_{DD}**		**74**	**G13**	**210**
	B11	**128**	**42**	**E0-CLK1**	**165**		**G14**	**211**
9	**B10**	**127**		**E1**	**166**	**75**	**G15 / TDO**	**212**
	B9	**126**	**43**	**E2**	**167**	**76**	**GND**	
10	**B8**	**125**		**E3**	**168**	**77**	**H0**	**213**
11	**B7**	**124**	**44**	**E4**	**169**		**H1**	**214**
	B6	**123**	**45**	**GND**		**78**	**H2**	**215**
12	**B5**	**122**	**46**	**E5**	**170**		**H3**	**216**
13	**GND**			**E6**	**171**	**79**	**H4**	**217**
14	**B4**	**121**	**47**	**E7**	**172**	**80**	**H5**	**218**
	B3	**120**	**48**	**E8**	**173**		**H6**	**219**
15	**B2**	**119**		**E9**	**174**	**81**	**H7**	**220**
	B1	**118**	**49**	**E10**	**175**	**82**	**H8**	**221**
16	**B0**	**117**		**E11**	**176**		**H9**	**222**
17	**C15 / TMS**	**148**	**50**	**E12**	**177**	**83**	**H10**	**223**
	C14	**147**	**51**	**E13**	**178**	**84**	**V_{DD}**	
18	**C13**	**146**		**E14**	**179**		**H11**	**224**
19	**C12**	**145**	**52**	**E15**	**180**	**85**	**H12**	**225**
20	**V_{DD}**		**53**	**V_{DD}**		**86**	**H13**	**226**
	C11	**144**	**54**	**F0**	**181**		**H14**	**227**
21	**C10**	**143**		**F1**	**182**	**87**	**H15**	**228**
	C9	**142**	**55**	**F2**	**183**	**88**	**GND**	
22	**C8**	**141**		**F3**	**184**	**89**	**IN0-CLK0**	
23	**C7**	**140**	**56**	**F4**	**185**	**90**	**IN2**	
	C6	**139**	**57**	**F5**	**186**	**91**	**IN1**	
24	**C5**	**138**		**F6**	**187**	**92**	**IN3**	
25	**C4**	**137**	**58**	**F7**	**188**	**93**	**V_{DD}**	
	C3	**136**	**59**	**F8**	**189**	**94**	**A15-CLK3**	**116**
26	**C2**	**135**		**F9**	**190**		**A14**	**115**
	C1	**134**	**60**	**F10**	**191**	**95**	**A13**	**114**
27	**C0**	**133**	**61**	**GND**		**96**	**A12**	**113**
28	**GND**			**F11**	**192**	**97**	**GND**	
29	**D15**	**164**	**62**	**F12**	**193**		**A11**	**112**
	D14	**163**	**63**	**F13**	**194**	**98**	**A10**	**111**
30	**D13**	**162**		**F14**	**195**		**A9**	**110**
31	**D12**	**161**	**64**	**F15 / TCK**	**196**	**99**	**A8**	**109**
	D11	**160**	**65**	**G0**	**197**	**100**	**A7**	**108**
32	**D10**	**159**		**G1**	**198**		**A6**	**107**
	D9	**158**	**66**	**G2**	**199**			
33	**D8**	**157**		**G3**	**200**			

PZ3128/PZ5128 128 Pin LQFP

PIN	NAME	NODE	PIN	NAME	NODE	PIN	NAME	NODE	PIN	NAME	NODE
1	**NC**			**C1**	**162**	**65**	**NC**		**96**	**H0**	**241**
2	**NC**		**33**	**C0**	**161**	**66**	**NC**			**H1**	**242**
3	**V_{DD}**		**34**	**D15**	**192**	**67**	**V_{DD}**		**97**	**H2**	**243**
4	**A4**	**133**		**D14**	**191**	**68**	**E13**	**206**	**98**	**H3**	**244**
5	**A3**	**132**	**35**	**D13**	**190**		**E14**	**207**	**99**	**H4**	**245**
6	**A2**	**131**	**36**	**GND**		**69**	**E15**	**208**	**100**	**GND**	
	A1	**130**	**37**	**NC**		**70**	**F0**	**209**	**101**	**NC**	
7	**A0**	**129**	**38**	**NC**			**F1**	**210**	**102**	**NC**	
8	**B15 / TDI**	**160**	**39**	**NC**		**71**	**F2**	**211**	**103**	**NC**	
	B14	**159**	**40**	**V_{DD}**		**72**	**F3**	**212**	**104**	**V_{DD}**	
9	**B13**	**158**	**41**	**D12**	**189**	**73**	**F4**	**213**	**105**	**H5**	**246**
10	**B12**	**157**	**42**	**D11**	**188**	**74**	**F5**	**214**		**H6**	**247**
11	**B11**	**156**	**43**	**D10**	**187**		**F6**	**215**	**106**	**H7**	**248**
12	**B10**	**155**		**D9**	**186**	**75**	**F7**	**216**	**107**	**H8**	**249**
	B9	**154**	**44**	**D8**	**185**	**76**	**F8**	**217**		**H9**	**250**
13	**B8**	**153**	**45**	**D7**	**184**		**F9**	**218**	**108**	**H10**	**251**
14	**B7**	**152**		**D6**	**183**	**77**	**F10**	**219**	**109**	**H11**	**252**
15	**GND**		**46**	**D5**	**182**	**78**	**F11**	**220**	**110**	**H12**	**253**
	B6	**151**	**47**	**D4**	**181**	**79**	**GND**		**111**	**H13**	**254**
16	**B5**	**150**	**48**	**D3**	**180**	**80**	**F12**	**221**		**H14**	**255**
17	**B4**	**149**	**49**	**D2**	**179**	**81**	**F13**	**222**	**112**	**H15**	**256**
18	**B3**	**148**		**D1**	**178**		**F14**	**223**	**113**	**GND**	
19	**B2**	**147**	**50**	**D0-CLK2**	**177**	**82**	**F15 / TCK**	**224**	**114**	**IN0-CLK0**	
	B1	**146**	**51**	**GND**		**83**	**G0**	**225**	**115**	**IN2**	
20	**B0**	**145**	**52**	**V_{DD}**			**G1**	**226**	**116**	**IN1**	
21	**C15 / TMS**	**176**	**53**	**E0-CLK1**	**193**	**84**	**G2**	**227**	**117**	**IN3**	
	C14	**175**		**E1**	**194**	**85**	**G3**	**228**	**118**	**V_{DD}**	
22	**C13**	**174**	**54**	**E2**	**195**	**86**	**G4**	**229**	**119**	**A15-CLK3**	**144**
23	**C12**	**173**	**55**	**E3**	**196**	**87**	**G5**	**230**		**A14**	**143**
24	**V_{DD}**		**56**	**E4**	**197**	**88**	**V_{DD}**		**120**	**A13**	**142**
25	**C11**	**172**	**57**	**E5**	**198**		**G6**	**231**	**121**	**A12**	**141**
26	**C10**	**171**		**E6**	**199**	**89**	**G7**	**232**	**122**	**A11**	**140**
	C9	**170**	**58**	**E7**	**200**	**90**	**G8**	**233**	**123**	**A10**	**139**
27	**C8**	**169**	**59**	**E8**	**201**		**G9**	**234**		**A9**	**138**
28	**C7**	**168**		**E9**	**202**	**91**	**G10**	**235**	**124**	**A8**	**137**
	C6	**167**	**60**	**E10**	**203**	**92**	**G11**	**236**	**125**	**A7**	**136**
29	**C5**	**166**	**61**	**E11**	**204**	**93**	**G12**	**237**		**A6**	**135**
30	**C4**	**165**	**62**	**E12**	**205**	**94**	**G13**	**238**	**126**	**A5**	**134**
31	**C3**	**164**	**63**	**GND**			**G14**	**239**	**127**	**GND**	
32	**C2**	**163**	**64**	**NC**		**95**	**G15 / TDO**	**240**	**128**	**NC**	

PZ3128/PZ5128 160 Pin PQFP

PIN	NAME	NODE	PIN	NAME	NODE	PIN	NAME	NODE	PIN	NAME	NODE
1	**NC**		**41**	**C0**	**193**	**81**	**NC**		**121**	**H0**	**273**
2	**NC**		**42**	**GND**		**82**	**NC**			**H1**	**274**
3	**NC**		**43**	**D15**	**224**	**83**	**NC**		**122**	**H2**	**275**
4	**NC**			**D14**	**223**	**84**	**NC**		**123**	**H3**	**276**
5	**NC**		**44**	**NC**		**85**	**NC**		**124**	**NC**	
6	**NC**		**45**	**NC**		**86**	**NC**		**125**	**NC**	
7	**NC**		**46**	**NC**		**87**	**NC**		**126**	**NC**	
8	**V_{DD}**		**47**	**NC**			**F1**	**242**	**127**	**NC**	
9	**B15 / TDI**	**192**	**48**	**D13**	**222**	**88**	**F2**	**243**	**128**	**H4**	**277**
	B14	**191**	**49**	**D12**	**221**	**89**	**F3**	**244**	**129**	**H5**	**278**
10	**B13**	**190**	**50**	**D11**	**220**	**90**	**F4**	**245**		**H6**	**279**
11	**B12**	**189**	**51**	**D10**	**219**	**91**	**F5**	**246**	**130**	**H7**	**280**
12	**B11**	**188**		**D9**	**218**		**F6**	**247**	**131**	**H8**	**281**
13	**B10**	**187**	**52**	**D8**	**217**	**92**	**F7**	**248**		**H9**	**282**
	B9	**186**	**53**	**D7**	**216**	**93**	**F8**	**249**	**132**	**H10**	**283**
14	**B8**	**185**		**D6**	**215**		**F9**	**250**	**133**	**V_{DD}**	
15	**B7**	**184**	**54**	**D5**	**214**	**94**	**F10**	**251**	**134**	**H11**	**284**
	B6	**183**	**55**	**V_{DD}**		**95**	**GND**		**135**	**H12**	**285**
16	**B5**	**182**	**56**	**D4**	**213**	**96**	**F11**	**252**	**136**	**H13**	**286**
17	**GND**		**57**	**D3**	**212**	**97**	**F12**	**253**		**H14**	**287**
18	**B4**	**181**	**58**	**D2**	**211**	**98**	**F13**	**254**	**137**	**H15**	**288**
19	**B3**	**180**		**D1**	**210**		**F14**	**255**	**138**	**GND**	
20	**B2**	**179**	**59**	**D0/CLK2**	**209**	**99**	**F15 / TCK**	**256**	**139**	**IN0/CLK0**	
	B1	**178**	**60**	**GND**		**100**	**G0**	**257**	**140**	**IN2**	
21	**B0**	**177**	**61**	**V_{DD}**			**G1**	**258**	**141**	**IN1**	
22	**C15 / TMS**	**208**	**62**	**E0/CLK1**	**225**	**101**	**G2**	**259**	**142**	**IN3**	
	C14	**207**		**E1**	**226**	**102**	**G3**	**260**	**143**	**V_{DD}**	
23	**C13**	**206**	**63**	**E2**	**227**	**103**	**G4**	**261**	**144**	**A15/CLK3**	**176**
24	**C12**	**205**	**64**	**E3**	**228**	**104**	**V_{DD}**			**A14**	**175**
25	**C11**	**204**	**65**	**E4**	**229**	**105**	**G5**	**262**	**145**	**A13**	**174**
26	**V_{DD}**		**66**	**GND**			**G6**	**263**	**146**	**A12**	**173**
27	**C10**	**203**	**67**	**E5**	**230**	**106**	**G7**	**264**	**147**	**A11**	**172**
	C9	**202**		**E6**	**231**	**107**	**G8**	**265**	**148**	**GND**	
28	**C8**	**201**	**68**	**E7**	**232**		**G9**	**266**	**149**	**A10**	**171**
29	**C7**	**200**	**69**	**E8**	**233**	**108**	**G10**	**267**		**A9**	**170**
	C6	**199**		**E9**	**234**	**109**	**G11**	**268**	**150**	**A8**	**169**
30	**C5**	**198**	**70**	**E10**	**235**	**110**	**G12**	**269**	**151**	**A7**	**168**
31	**C4**	**197**	**71**	**E11**	**236**	**111**	**G13**	**270**		**A6**	**167**
32	**C3**	**196**	**72**	**E12**	**237**		**G14**	**271**	**152**	**A5**	**166**
33	**C2**	**195**	**73**	**E13**	**238**	**112**	**G15 / TDO**	**272**	**153**	**A4**	**165**
	C1	**194**	**74**	**NC**		**113**	**GND**		**154**	**NC**	
34	**NC**		**75**	**NC**		**114**	**NC**		**155**	**NC**	
35	**NC**		**76**	**NC**		**115**	**NC**		**156**	**NC**	
36	**NC**		**77**	**NC**		**116**	**NC**		**157**	**NC**	
37	**NC**			**E14**	**239**	**117**	**NC**		**158**	**A3**	**164**
38	**NC**		**78**	**E15**	**240**	**118**	**NC**		**159**	**A2**	**163**
39	**NC**		**79**	**V_{DD}**		**119**	**NC**			**A1**	**162**
40	**NC**		**80**	**F0**	**241**	**120**	**NC**		**160**	**A0**	**161**

Appendix C *Design Examples*

This Appendix contains a wide variety of design examples. The purpose of this Appendix is to give the reader reference designs which can be used as is, or can be modified to meet the individual designer's needs. An electronic version of all of these design files (.phd) along with their corresponding simulation files (.scl) are loaded automatically when the XPLA Designer software is installed.

This section contains the following Design Examples:

Module	Electronic Directory	Module Title
_8bshift	xpla\examples\8bshift	8-Bit Shift Register
adder	xpla\examples\adders\adder	N-Bit Adder
adder8	xpla\examples\adders\adder8	8-Bit Adder - High Level
adder8_d	xpla\examples\adders\adder8_d	8-Bit Adder - Low Level
addr4b	xpla\examples\adders\addr4b	4-Bit Adder w/ Carry In and Out
bcd7	xpla\examples\bcd	BCD to 7 Segment Decoder
bidirect	xpla\examples\bidirect	Bidirectional IOs
comp8	xpla\examples\comp8	8-Bit Equality Comparator
cnt16_d	xpla\examples\counters\cnt16_d	16-Bit Counter - Low Level
cntr2_16	xpla\examples\counters\cntr2_16	Dual 16-Bit Counters
demo	xpla\examples\counters\demo	3-Bit Counter
gcnt16	xpla\examples\counters\gcnt_d	16-Bit Gray Code Counter
gray4	xpla\examples\counters\gray4	4-Bit Gray Code Counter
p2	xpla\examples\counters\p2	Timer/Counter
p7	xpla\examples\counters\p7	16-Bit Loadable Binary Counter
p8	xpla\examples\counters\p8	16-Bit Synchronous Prescaled Counter
crc8s	xpla\examples\crc8s	Serial CRC Generator
dec38	xpla\examples\dec38	3 to 8 Decoder
dr8	xpla\examples\dr8	8-Bit Loadable Data Register
dram_ctl	xpla\examples\dram_ctl	DRAM Controller
m16_8	xpla\examples\m16_8	16 to 8 Multiplexer
m41_2	xpla\examples\m41_2	Dual 4 to 1 Multiplexer
p1	xpla\examples\p1	Datapath Circuit
p3	xpla\examples\p3	Small State Machine
p5	xpla\examples\p5	Arithmetic Circuit
p6	xpla\examples\p6	1-Bit Accumulator
p9	xpla\examples\p9	Address Decoder
parity	xpla\examples\parity	8-Bit Fast Parity Generator
refresh	xpla\examples\refresh	DRAM Refresh Counter
tbird	xpla\examples\tbird	Thunderbird Taillight Control Circuit
traffic1	xpla\examples\traffic1	Denver Internatiomal Air Traffic Control

8-Bit Shift Register

```
Module _8bshift

"bwb"

Title   '8 bit shift register'

        a,b                         pin; "Data to shift in
        Clear, Clock                pin; "Shifter clock and clear signal setup
        qa,qb,qc,qd,qe,qf,qg,qh     pin istype 'reg,buffer'; "8 shifter bits


equations

        qa :=a & b; "Data into first bit is the AND result of 'a' and 'b'
        [qb,qc,qd,qe,qf,qg,qh] := [qa,qb,qc,qd,qe,qf,qg]; " shifts data one bit at each clock

        [qa,qb,qc,qd,qe,qf,qg,qh].clk = Clock; "sets up clock
        [qa,qb,qc,qd,qe,qf,qg,qh].ar = !Clear; "sets up clear


end
```

N-Bit Adder

```
MODULE adder

TITLE ' N bit Adder example'

"bwb

        "The number of bits added can be ajusted by setting the appropriate number
        "of bits in 'a', 'b', and 'c'.

"Inputs
    CLK,RST,OENAB               PIN 43,1,2; "defines clock, reset, and output enable
      A_3..A_0                  PIN;        "sets four bit 'a' input
      B_3..B_0                  PIN;        "sets four bit 'b' input

"Outputs

      C_3..C_0              PIN ISTYPE 'REG';   "sets four bit 'c' output


      A_IN  = [A_3..A_0];     "combines the four 'a' inputs into one variable
      B_IN  = [B_3..B_0];     "combines the four 'b' inputs into one variable
      C_OUT = [C_3..C_0];   "combines the four 'c' outputs into one variable



Equations

      C_OUT.OE  = !OENAB;       " Defines output enable to be active low

      C_OUT.CLK = CLK;          "connects clock to all four output registers
      C_OUT.AR  = RST;          "connects reset to all four output registers


"EQUATIONS FOR ADDER

      C_OUT := A_IN + B_IN;     "sums the 'a' and 'b' inputs into 'c' at the clock pulse

END
```

8-Bit Adder - High Level

```
MODULE adder8

TITLE '8-Bit Adder '

        A7..A0                  PIN;
        B7..B0                  PIN;

        S7..S0                  PIN;
        CO                      PIN;

        A               = [0,A7..A0];
        B               = [0,B7..B0];
        SUM             = [CO,S7..S0];

EQUATIONS
        SUM             = A + B;

END
```

8-Bit Adder - Low Level

```
module adder8_d ;

" low level implementation of an adder

a7..a0 pin ;
b7..b0 pin ;

c7..c1 node istype 'com' ;
s7..s0 pin istype 'com' ;
c_out pin istype 'com' ;

equations

s0 = a0 $ b0 ;
c1 = a0 & b0 ;

s1 = a1 $ b1 $ c1 ;
c2 = a1 & b1 # (a1 $ b1) & c1 ;

s2 = a2 $ b2 $ c2 ;
c3 = a2 & b2 # (a2 $ b2) & c2 ;

s3 = a3 $ b3 $ c3 ;
c4 = a3 & b3  # (a3 $ b3) & c3 ;

s4 = a4 $ b4 $ c4 ;
c5 = a4 & b4 # (a4 & b4) & c4 ;

s5 = a5 $ b5 $ c5 ;
c6 = a5 & b5 # (a5 $ b5) & c5 ;

s6 = a6 $ b6 $ c6 ;
c7 = a6 & b6 # (a6 $ b6) & c6 ;

s7 = a7 $ b7 $ c7 ;
c_out = a7 & b7 # (a7 $ b7) & c7 ;

end
```

4-Bit Adder with Carry in and Carry out

```
Module  addr4b

Title   'Four bit adder with carry in and carry out'

clk        pin;
a3..a0   pin;
b3..b0   pin;
sum3..sum0 pin istype 'reg,buffer';
carryin pin;
carryout pin istype 'reg,buffer';

a = [0,a3..a0];
b = [0,b3..b0];
sum = [carryout,sum3..sum0];

equations

sum.c = clk;

sum := a + b + carryin;

end
```

BCD to 7 Segment Decoder

```
Module  BCD7
"bwb
Title   'BCD to 7 segment decoder'

" Seven segment display decoder/driver with active low outputs
"
"                       Segments:       -a-
"                                     f|   |b
"                                       -g-
"                                     e|   |c
"                                       -d-
"

 D3..D0                 PIN ;                   "BCD INPUT
 A,B,C,D,E,F,G          PIN ISTYPE 'COM';       "SEGMENT OUTPUTS
 OE                     PIN ;                   "OUTPUT ENABLE

 BCD    =[D3..D0];
 LED    =[A,B,C,D,E,F,G];
 ON,OFF         = 0,1;                          "INVERTED SENSE FOR COMMON ANODE
LED'S

equations

  LED.OE = !OE;                                         "DEFINE OUTPUT ENABLE

truth_table(BCD  ->[  A ,  B ,  C ,  D ,  E ,  F ,  G  ])
              0   ->[ ON , ON , ON , ON , ON , ON , OFF ];
              1   ->[ OFF, ON , ON , OFF, OFF, OFF, OFF ];
              2   ->[ ON , ON , OFF, ON , ON , OFF, ON  ];
              3   ->[ ON , ON , ON , ON , OFF, OFF, ON  ];
              4   ->[ OFF, ON , ON , OFF, OFF, ON , ON  ];
              5   ->[ ON , OFF, ON , ON , OFF, ON , ON  ];
              6   ->[ ON , OFF, ON , ON , ON , ON , ON  ];
              7   ->[ ON , ON , ON , OFF, OFF, OFF, OFF ];
              8   ->[ ON , ON , ON , ON , ON , ON , ON  ];
              9   ->[ ON , ON , ON , OFF, OFF, ON , ON  ];
             10   ->[ ON , ON , ON , ON , ON , OFF, ON  ];
             11   ->[ OFF, OFF, ON , ON , ON , ON , ON  ];
             12   ->[ ON , OFF, OFF, ON , ON , ON , OFF ];
             13   ->[ OFF, ON , ON , ON , ON , OFF, ON  ];
             14   ->[ ON , OFF, OFF, ON , ON , ON , ON  ];
             15   ->[ ON , OFF, OFF, OFF, ON , ON , ON  ];

end
```

Bidirectional I/O's

```
MODULE bidirect

TITLE 'Bidirectional I/Os'

a3..a0 pin istype 'reg' ;
b3..b0 pin ;
c3..c0 pin ;
d3..d0 pin ;

clk pin ;
ena pin ;
enb pin ;

a = [a3..a0] ;
b = [b3..b0] ;
c = [c3..c0] ;
d = [d3..d0] ;

equations

a := c ;
b  = d ;

a.clk = clk ;
a.oe = ena ;
b.oe = enb ;

END
```

8-bit Equality Comparator - High Level Implementation

```
MODULE comp8

TITLE '8-bit equality comparator - high level implementation'

a7..a0 pin ;
b7..b0 pin ;
eq pin ;

a = [a7..a0] ;
b = [b7..b0] ;

equations

eq = (a==b) ;

END
```

16-Bit Counter - Low Level

```
module cnt16_d ;

“ This is the low -level implementation of a 16-bit counter
"Inputs
                                        pin ;
    D15,D14,D13,D12,D11,D10,D9,D8       pin ;
    D7,D6,D5,D4,D3,D2,D1,D0             pin ;
"Outputs
    Q15,Q14,Q13,Q12,Q11,Q10,Q9,Q8       pin istype 'buffer';
    Q7,Q6,Q5,Q4,Q3,Q2,Q1,Q0             pin istype 'buffer';


    DIN  = [D15,D14,D13,D12,D11,D10,D9,D8,D7,D6,D5,D4,D3,D2,D1,D0];
    QOUT = [Q15,Q14,Q13,Q12,Q11,Q10,Q9,Q8,Q7,Q6,Q5,Q4,Q3,Q2,Q1,Q0];

    H,L,C,X = 1,0,.C.,.X.;

Equations

    QOUT.CLK = CLK;
    QOUT.AR  = RST;

Q15.T =Q14 & Q13 & Q12 & Q11 & Q10 & Q9 & Q8 & Q7 & Q6 & Q5 & Q4 & Q3 &
Q2 & Q1 & Q0 ;
Q14.T  = Q13 & Q12 & Q11 & Q10 & Q9 & Q8 & Q7 & Q6 & Q5 & Q4 & Q3 & Q2 &
Q1 & Q0 ;
Q13.T  = Q12 & Q11 & Q10 & Q9 & Q8 & Q7 & Q6 & Q5 & Q4 & Q3 & Q2 & Q1 &
Q0 ;
Q12.T  = Q11 & Q10 & Q9 & Q8 & Q7 & Q6 & Q5 & Q4 & Q3 & Q2 & Q1 & Q0 ;
Q11.T  = Q10 & Q9 & Q8 & Q7 & Q6 & Q5 & Q4 & Q3 & Q2 & Q1 & Q0 ;
Q10.T  = Q9 & Q8 & Q7 & Q6 & Q5 & Q4 & Q3 & Q2 & Q1 & Q0 ;
Q9.T  = Q8 & Q7 & Q6 & Q5 & Q4 & Q3 & Q2 & Q1 & Q0 ;
Q8.T  = Q7 & Q6 & Q5 & Q4 & Q3 & Q2 & Q1 & Q0 ;
Q7.T  = Q6 & Q5 & Q4 & Q3 & Q2 & Q1 & Q0 ;
Q6.T  = Q5 & Q4 & Q3 & Q2 & Q1 & Q0 ;
Q5.T  = Q4 & Q3 & Q2 & Q1 & Q0 ;
Q4.T  = Q3 & Q2 & Q1 & Q0 ;
Q3.T  = Q2 & Q1 & Q0 ;
Q2.T  = Q1 & Q0 ;
Q1.T  = Q0 ;
Q0.T  = 1;

end
```

Dual 16-Bit Up/Down/Loadable/Enabled/Resetable Counters'

```
MODULE cntr2_16

title '2 16 bit up/down/loadable/enabled/resetable counters'

"NOTE: reset is load & count_enab & dir."
"NOTE: BIDIR pins are used for the load of each counter"

DECLARATIONS

dir             pin 1;
load            pin 2;

count_enab      pin 44;
clk             pin 43;

ca15..ca0       pin 4,5,6,7,8,9,11,12,13,14,16,17,18,19,20,21 istype 'reg';
cb15..cb0       pin 41,40,39,38,37,36,34,33,32,31,29,28,27,26,25,24 istype 'reg';

ca = [ca15..ca0];
cb = [cb15..cb0];

EQUATIONS

ca.clk = clk;
cb.clk = clk;

ca.ar = load & count_enab & dir;
cb.ar = load & count_enab & dir;

ca.oe = !load;
cb.oe = !load;

when (load == 1) then
        {
                ca.d := ca.pin;
                cb.d := cb.pin;
        }
else
{
        when (count_enab == 1) then
        {
                when (dir == 0) then
                {
                        ca.d := ca.q + 1;
                        cb.d := cb.q + 1;
                }
```

```
                else when (dir == 1) then
                {
                        ca.d := ca.q - 1;
                        cb.d := cb.q - 1;
                }
        }
        else
        {
                ca.d = ca.q;
                cb.d = cb.q;
        }
}


END
```

3-Bit Counter

```
Module  DEMO
Title   'My first design:3-bit counter'

CLOCK, RESET          pin;
bit2..bit0            pin istype 'reg';
count = [bit2..bit0];

equations

count.CLK = CLOCK;
count.AR = RESET;
count = count.Q + 1;

end;
```

16-Bit Gray Code Counter

```
module gcnt16 ;

title ' 16-bit gray code counter'

q15..q0 pin istype 'reg' ;

clk pin ;
rst pin ;

count = [q15..q0] ;

equations

count.ar = rst ;
count.clk = clk ;

q15 := q15 & q14 & !q13 & !q12 & !q11 & !q10 & !q9 & !q8 & !q7 & !q6
        & !q5 & !q4 & !q3 & !q2 & !q1 & !q0
# q15 & q14 & q13 & q12 & !q11 & !q10 & !q9 & !q8 & !q7 & !q6
        & !q5 & !q4 & !q3 & !q2 & !q1 & !q0
# q15 & !q14 & q13 & !q12 & !q11 & !q10 & !q9 & !q8 & !q7 & !q6
        & !q5 & !q4 & !q3 & !q2 & !q1 & !q0
# q15 & !q14 & !q13 & !q12 & !q11 & !q10 & !q9 & !q8 & !q7 & !q6
        & !q5 & !q4 & !q3 & !q2 & !q1 & !q0 ;

q14 := !q15 & q14 & q13 & !q12 & !q11 & !q10 & !q9 & !q8 & !q7 & !q6
        & !q5 & !q4 & !q3 & !q2 & !q1 & !q0
# !q15 & q14 & q13 & q12 & !q11 & !q10 & !q9 & !q8 & !q7 & !q6
        & !q5 & !q4 & !q3 & !q2 & !q1 & !q0
# !q15 & q14 & !q13 & q12 & !q11 & !q10 & !q9 & !q8 & !q7 & !q6
        & !q5 & !q4 & !q3 & !q2 & !q1 & !q0
# !q15 & q14 & !q13 & !q12 & !q11 & !q10 & !q9 & !q8 & !q7 & !q6
        & !q5 & !q4 & !q3 & !q2 & !q1 & !q0
# q15 & q14 & !q13 & !q12 & !q11 & !q10 & !q9 & !q8 & !q7 & !q6
        & !q5 & !q4 & !q3 & !q2 & !q1 & !q0
# q15 & q14 & q13 & q12 & !q11 & !q10 & !q9 & !q8 & !q7 & !q6
        & !q5 & !q4 & !q3 & !q2 & !q1 & !q0 ;

q13 := !q15 & !q14 & !q13 & q12 & !q11 & !q10 & !q9 & !q8 & !q7 & !q6
        & !q5 & !q4 & !q3 & !q2 & !q1 & !q0
# !q15 & !q14 & q13 & q12 & !q11 & !q10 & !q9 & !q8 & !q7 & !q6
        & !q5 & !q4 & !q3 & !q2 & !q1 & !q0
# !q15 & !q14 & q13 & q12 & q11 & !q10 & !q9 & !q8 & !q7 & !q6
        & !q5 & !q4 & !q3 & !q2 & !q1 & !q0
# !q15 & !q14 & q13 & !q12 & q11 & !q10 & !q9 & !q8 & !q7 & !q6
        & !q5 & !q4 & !q3 & !q2 & !q1 & !q0
```

```
# !q15 & !q14 & q13 & !q12 & !q11 & !q10 & !q9 & !q8 & !q7 & !q6
        & !q5 & !q4 & !q3 & !q2 & !q1 & !q0
# !q15 & q14 & q13 & !q12 & !q11 & !q10 & !q9 & !q8 & !q7 & !q6
        & !q5 & !q4 & !q3 & !q2 & !q1 & !q0
# q15 & q14 & !q13 & !q12 & !q11 & !q10 & !q9 & !q8 & !q7 & !q6
        & !q5 & !q4 & !q3 & !q2 & !q1 & !q0
# q15 & q14 & q13 & !q12 & !q11 & !q10 & !q9 & !q8 & !q7 & !q6
        & !q5 & !q4 & !q3 & !q2 & !q1 & !q0 ;

q12 := !q15 & !q14 & !q13 & !q12 & q11 & !q10 & !q9 & !q8 & !q7 & !q6
        & !q5 & !q4 & !q3 & !q2 & !q1 & !q0
# !q15 & !q14 & !q13 & q12 & q11 & !q10 & !q9 & !q8 & !q7 & !q6
        & !q5 & !q4 & !q3 & !q2 & !q1 & !q0
# !q15 & !q14 & !q13 & q12 & q11 & q10 & !q9 & !q8 & !q7 & !q6
        & !q5 & !q4 & !q3 & !q2 & !q1 & !q0
# !q15 & !q14 & !q13 & q12 & !q11 & q10 & !q9 & !q8 & !q7 & !q6
        & !q5 & !q4 & !q3 & !q2 & !q1 & !q0
# !q15 & !q14 & !q13 & q12 & !q11 & !q10 & !q9 & !q8 & !q7 & !q6
        & !q5 & !q4 & !q3 & !q2 & !q1 & !q0
# !q15 & !q14 & q13 & q12 & !q11 & !q10 & !q9 & !q8 & !q7 & !q6
        & !q5 & !q4 & !q3 & !q2 & !q1 & !q0
# !q15 & q14 & q13 & !q12 & !q11 & !q10 & !q9 & !q8 & !q7 & !q6
        & !q5 & !q4 & !q3 & !q2 & !q1 & !q0
# !q15 & q14 & q13 & q12 & !q11 & !q10 & !q9 & !q8 & !q7 & !q6
        & !q5 & !q4 & !q3 & !q2 & !q1 & !q0 ;

q11 := !q15 & !q14 & !q13 & !q12 & !q11 & q10 & !q9 & !q8 & !q7 & !q6
        & !q5 & !q4 & !q3 & !q2 & !q1 & !q0
# !q15 & !q14 & !q13 & !q12 & q11 & q10 & !q9 & !q8 & !q7 & !q6
        & !q5 & !q4 & !q3 & !q2 & !q1 & !q0
# !q15 & !q14 & !q13 & !q12 & q11 & q10 & q9 & !q8 & !q7 & !q6
        & !q5 & !q4 & !q3 & !q2 & !q1 & !q0
# !q15 & !q14 & !q13 & !q12 & q11 & !q10 & q9 & !q8 & !q7 & !q6
        & !q5 & !q4 & !q3 & !q2 & !q1 & !q0
# !q15 & !q14 & !q13 & !q12 & q11 & !q10 & !q9 & !q8 & !q7 & !q6
        & !q5 & !q4 & !q3 & !q2 & !q1 & !q0
# !q15 & !q14 & !q13 & q12 & q11 & !q10 & !q9 & !q8 & !q7 & !q6
        & !q5 & !q4 & !q3 & !q2 & !q1 & !q0
# !q15 & !q14 & q13 & q12 & !q11 & !q10 & !q9 & q8 & !q7 & !q6
        & !q5 & !q4 & !q3 & !q2 & !q1 & !q0
# !q15 & !q14 & q13 & q12 & q11 & !q10 & !q9 & !q8 & !q7 & !q6
        & !q5 & !q4 & !q3 & !q2 & !q1 & !q0 ;

q10 := !q15 & !q14 & !q13 & !q12 & !q11 & !q10 & q9 & !q8 & !q7 & !q6
        & !q5 & !q4 & !q3 & !q2 & !q1 & !q0
# !q15 & !q14 & !q13 & !q12 & !q11 & q10 & q9 & !q8 & !q7 & !q6
        & !q5 & !q4 & !q3 & !q2 & !q1 & !q0
# !q15 & !q14 & !q13 & !q12 & !q11 & q10 & q9 & q8 & !q7 & !q6
```

```
          & !q5 & !q4 & !q3 & !q2 & !q1 & !q0
# !q15 & !q14 & !q13 & !q12 & !q11 & q10 & !q9 & q8 & !q7 & !q6
          & !q5 & !q4 & !q3 & !q2 & !q1 & !q0
# !q15 & !q14 & !q13 & !q12 & !q11 & q10 & !q9 & !q8 & !q7 & !q6
          & !q5 & !q4 & !q3 & !q2 & !q1 & !q0
# !q15 & !q14 & !q13 & !q12 & q11 & q10 & !q9 & !q8 & !q7 & !q6
          & !q5 & !q4 & !q3 & !q2 & !q1 & !q0
# !q15 & !q14 & !q13 & q12 & q11 & !q10 & !q9 & !q8 & !q7 & !q6
          & !q5 & !q4 & !q3 & !q2 & !q1 & !q0
# !q15 & !q14 & !q13 & q12 & q11 & q10 & !q9 & !q8 & !q7 & !q6
          & !q5 & !q4 & !q3 & !q2 & !q1 & !q0 ;

q9 := !q15 & !q14 & !q13 & !q12 & !q11 & !q10 & !q9 & q8 & !q7 & !q6
          & !q5 & !q4 & !q3 & !q2 & !q1 & !q0
# !q15 & !q14 & !q13 & !q12 & !q11 & !q10 & q9 & q8 & !q7 & !q6
          & !q5 & !q4 & !q3 & !q2 & !q1 & !q0
# !q15 & !q14 & !q13 & !q12 & !q11 & !q10 & q9 & q8 & q7 & !q6
          & !q5 & !q4 & !q3 & !q2 & !q1 & !q0
# !q15 & !q14 & !q13 & !q12 & !q11 & !q10 & q9 & !q8 & q7 & !q6
          & !q5 & !q4 & !q3 & !q2 & !q1 & !q0
# !q15 & !q14 & !q13 & !q12 & !q11 & !q10 & q9 & !q8 & !q7 & !q6
          & !q5 & !q4 & !q3 & !q2 & !q1 & !q0
# !q15 & !q14 & !q13 & !q12 & !q11 & q10 & q9 & !q8 & !q7 & !q6
          & !q5 & !q4 & !q3 & !q2 & !q1 & !q0
# !q15 & !q14 & !q13 & !q12 & q11 & q10 & !q9 & !q8 & !q7 & !q6
          & !q5 & !q4 & !q3 & !q2 & !q1 & !q0
# !q15 & !q14 & !q13 & !q12 & q11 & q10 & q9 & !q8 & !q7 & !q6
          & !q5 & !q4 & !q3 & !q2 & !q1 & !q0 ;

q8 := !q15 & !q14 & !q13 & !q12 & !q11 & !q10 & !q9 & !q8 & q7 & !q6
          & !q5 & !q4 & !q3 & !q2 & !q1 & !q0
# !q15 & !q14 & !q13 & !q12 & !q11 & !q10 & !q9 & q8 & q7 & !q6
          & !q5 & !q4 & !q3 & !q2 & !q1 & !q0
# !q15 & !q14 & !q13 & !q12 & !q11 & !q10 & !q9 & q8 & q7 & q6
          & !q5 & !q4 & !q3 & !q2 & !q1 & !q0
# !q15 & !q14 & !q13 & !q12 & !q11 & q10 & !q9 & q8 & !q7 & q6
          & !q5 & !q4 & !q3 & !q2 & !q1 & !q0
# !q15 & !q14 & !q13 & !q12 & !q11 & !q10 & !q9 & q8 & !q7 & !q6
          & !q5 & !q4 & !q3 & !q2 & !q1 & !q0
# !q15 & !q14 & !q13 & !q12 & !q11 & !q10 & q9 & q8 & !q7 & !q6
          & !q5 & !q4 & !q3 & !q2 & !q1 & !q0
# !q15 & !q14 & !q13 & !q12 & !q11 & q10 & q9 & !q8 & !q7 & !q6
          & !q5 & !q4 & !q3 & !q2 & !q1 & !q0
# !q15 & !q14 & !q13 & !q12 & !q11 & q10 & q9 & q8 & !q7 & !q6
          & !q5 & !q4 & !q3 & !q2 & !q1 & !q0 ;


q7 := !q15 & !q14 & !q13 & !q12 & !q11 & !q10 & !q9 & !q8 & !q7 & q6
```

```
          & !q5 & !q4 & !q3 & !q2 & !q1 & !q0
# !q15 & !q14 & !q13 & !q12 & !q11 & !q10 & !q9 & !q8 & q7 & q6
          & !q5 & q4 & !q3 & !q2 & !q1 & !q0
# !q15 & !q14 & !q13 & !q12 & !q11 & !q10 & !q9 & !q8 & q7 & q6
          & q5 & !q4 & !q3 & !q2 & !q1 & !q0
# !q15 & !q14 & !q13 & !q12 & !q11 & !q10 & !q9 & !q8 & q7 & !q6
          & q5 & !q4 & !q3 & !q2 & !q1 & !q0
# !q15 & !q14 & !q13 & !q12 & !q11 & !q10 & !q9 & !q8 & q7 & !q6
          & !q5 & !q4 & !q3 & !q2 & !q1 & !q0
# !q15 & !q14 & !q13 & !q12 & !q11 & !q10 & !q9 & q8 & q7 & !q6
          & !q5 & !q4 & !q3 & !q2 & !q1 & !q0
# !q15 & !q14 & !q13 & !q12 & !q11 & !q10 & q9 & q8 & !q7 & !q6
          & !q5 & !q4 & !q3 & !q2 & !q1 & !q0
# !q15 & !q14 & !q13 & !q12 & !q11 & !q10 & q9 & q8 & q7 & !q6
          & !q5 & !q4 & !q3 & !q2 & !q1 & !q0 ;

q6 := !q15 & !q14 & !q13 & !q12 & !q11 & !q10 & !q9 & !q8 & !q7 & !q6
          & q5 & !q4 & !q3 & !q2 & !q1 & !q0
# !q15 & !q14 & !q13 & !q12 & !q11 & !q10 & !q9 & !q8 & !q7 & q6
          & q5 & !q4 & !q3 & !q2 & !q1 & !q0
# !q15 & !q14 & !q13 & !q12 & !q11 & !q10 & !q9 & !q8 & !q7 & q6
          & q5 & q4 & !q3 & !q2 & !q1 & !q0
# !q15 & !q14 & !q13 & !q12 & !q11 & !q10 & !q9 & !q8 & !q7 & q6
          & !q5 & q4 & !q3 & !q2 & !q1 & !q0
# !q15 & !q14 & !q13 & !q12 & !q11 & !q10 & !q9 & !q8 & !q7 & q6
          & !q5 & !q4 & !q3 & !q2 & !q1 & !q0
# !q15 & !q14 & !q13 & !q12 & !q11 & !q10 & !q9 & !q8 & q7 & q6
          & !q5 & !q4 & !q3 & !q2 & !q1 & !q0
# !q15 & !q14 & !q13 & !q12 & !q11 & !q10 & !q9 & q8 & q7 & !q6
          & !q5 & !q4 & !q3 & !q2 & !q1 & !q0
# !q15 & !q14 & !q13 & !q12 & !q11 & !q10 & !q9 & q8 & q7 & q6
          & !q5 & !q4 & !q3 & !q2 & !q1 & !q0 ;

q5 := !q15 & !q14 & !q13 & !q12 & !q11 & !q10 & !q9 & !q8 & !q7 & !q6
          & !q5 & q4 & !q3 & !q2 & !q1 & !q0
# !q15 & !q14 & !q13 & !q12 & !q11 & !q10 & !q9 & !q8 & !q7 & !q6
          & q5 & q4 & !q3 & !q2 & !q1 & !q0
# !q15 & !q14 & !q13 & !q12 & !q11 & !q10 & !q9 & !q8 & !q7 & !q6
          & q5 & q4 & q3 & !q2 & !q1 & !q0
# !q15 & !q14 & !q13 & !q12 & !q11 & !q10 & !q9 & !q8 & !q7 & !q6
          & q5 & !q4 & q3 & !q2 & !q1 & !q0
# !q15 & !q14 & !q13 & !q12 & !q11 & !q10 & !q9 & !q8 & !q7 & !q6
          & q5 & !q4 & !q3 & !q2 & !q1 & !q0
# !q15 & !q14 & !q13 & !q12 & !q11 & !q10 & !q9 & !q8 & !q7 & q6
          & q5 & !q4 & !q3 & !q2 & !q1 & !q0
# !q15 & !q14 & !q13 & !q12 & !q11 & !q10 & !q9 & !q8 & q7 & q6
          & !q5 & !q4 & !q3 & !q2 & !q1 & !q0
# !q15 & !q14 & !q13 & !q12 & !q11 & !q10 & !q9 & !q8 & q7 & q6
```

```
        & q5 & !q4 & !q3 & !q2 & !q1 & !q0 ;

q4 := !q15 & !q14 & !q13 & !q12 & !q11 & !q10 & !q9 & !q8 & !q7 & !q6
        & !q5 & !q4 & q3 & !q2 & !q1 & !q0
# !q15 & !q14 & !q13 & !q12 & !q11 & !q10 & !q9 & !q8 & !q7 & !q6
        & !q5 & q4 & q3 & !q2 & !q1 & !q0
# !q15 & !q14 & !q13 & !q12 & !q11 & !q10 & !q9 & !q8 & !q7 & !q6
        & !q5 & q4 & q3 & q2 & !q1 & !q0
# !q15 & !q14 & !q13 & !q12 & !q11 & !q10 & !q9 & !q8 & !q7 & !q6
        & !q5 & q4 & !q3 & q2 & !q1 & !q0
# !q15 & !q14 & !q13 & !q12 & !q11 & !q10 & !q9 & !q8 & !q7 & !q6
        & !q5 & q4 & !q3 & !q2 & !q1 & !q0
# !q15 & !q14 & !q13 & !q12 & !q11 & !q10 & !q9 & !q8 & !q7 & !q6
        & q5 & q4 & !q3 & !q2 & !q1 & !q0
# !q15 & !q14 & !q13 & !q12 & !q11 & !q10 & !q9 & !q8 & !q7 & q6
        & q5 & !q4 & !q3 & !q2 & !q1 & !q0
# !q15 & !q14 & !q13 & !q12 & !q11 & !q10 & !q9 & !q8 & !q7 & q6
        & q5 & q4 & !q3 & !q2 & !q1 & !q0 ;

q3 := !q15 & !q14 & !q13 & !q12 & !q11 & !q10 & !q9 & !q8 & !q7 & !q6
        & !q5 & !q4 & !q3 & q2 & !q1 & !q0
# !q15 & !q14 & !q13 & !q12 & !q11 & !q10 & !q9 & !q8 & !q7 & !q6
        & !q5 & !q4 & q3 & q2 & !q1 & !q0
# !q15 & !q14 & !q13 & !q12 & !q11 & !q10 & !q9 & !q8 & !q7 & !q6
        & !q5 & !q4 & q3 & q2 & q1 & !q0
# !q15 & !q14 & !q13 & !q12 & !q11 & !q10 & !q9 & !q8 & !q7 & !q6
        & !q5 & !q4 & q3 & !q2 & q1 & !q0
# !q15 & !q14 & !q13 & !q12 & !q11 & !q10 & !q9 & !q8 & !q7 & !q6
        & !q5 & !q4 & q3 & !q2 & !q1 & !q0
# !q15 & !q14 & !q13 & !q12 & !q11 & !q10 & !q9 & !q8 & !q7 & !q6
        & !q5 & q4 & q3 & !q2 & !q1 & !q0
# !q15 & !q14 & !q13 & !q12 & !q11 & !q10 & q9 & !q8 & !q7 & !q6
        & q5 & q4 & !q3 & !q2 & !q1 & !q0
# !q15 & !q14 & !q13 & !q12 & !q11 & !q10 & !q9 & !q8 & !q7 & !q6
        & q5 & q4 & q3 & !q2 & !q1 & !q0 ;

q2 := !q15 & !q14 & !q13 & !q12 & !q11 & !q10 & !q9 & !q8 & !q7 & !q6
        & !q5 & !q4 & !q3 & !q2 & q1 & !q0
# !q15 & !q14 & !q13 & !q12 & !q11 & !q10 & !q9 & !q8 & !q7 & !q6
        & !q5 & !q4 & !q3 & q2 & q1 & !q0
# !q15 & !q14 & !q13 & !q12 & !q11 & !q10 & !q9 & !q8 & !q7 & !q6
        & !q5 & !q4 & !q3 & q2 & q1 & q0
# !q15 & !q14 & !q13 & !q12 & !q11 & !q10 & !q9 & !q8 & !q7 & !q6
        & !q5 & !q4 & !q3 & q2 & !q1 & q0
# !q15 & !q14 & !q13 & !q12 & !q11 & !q10 & !q9 & !q8 & !q7 & !q6
        & !q5 & !q4 & !q3 & q2 & !q1 & !q0
# !q15 & !q14 & !q13 & !q12 & !q11 & !q10 & !q9 & !q8 & !q7 & !q6
        & !q5 & !q4 & q3 & q2 & !q1 & !q0
```

```
# !q15 & !q14 & !q13 & !q12 & !q11 & !q10 & q9 & !q8 & !q7 & !q6
        & !q5 & q4 & q3 & !q2 & !q1 & !q0
# !q15 & !q14 & !q13 & !q12 & !q11 & !q10 & !q9 & !q8 & !q7 & !q6
        & !q5 & q4 & q3 & q2 & !q1 & !q0 ;

q1 := !q15 & !q14 & !q13 & !q12 & !q11 & !q10 & !q9 & !q8 & !q7 & !q6
        & !q5 & !q4 & !q3 & !q2 & !q1 & q0
# !q15 & !q14 & !q13 & !q12 & !q11 & !q10 & !q9 & !q8 & !q7 & !q6
        & !q5 & !q4 & !q3 & !q2 & q1 & q0
# !q15 & !q14 & !q13 & !q12 & !q11 & !q10 & !q9 & !q8 & !q7 & !q6
        & !q5 & !q4 & !q3 & !q2 & q1 & !q0
# !q15 & !q14 & !q13 & !q12 & !q11 & !q10 & !q9 & !q8 & !q7 & !q6
        & !q5 & !q4 & !q3 & q2 & q1 & !q0
# !q15 & !q14 & !q13 & !q12 & !q11 & !q10 & !q9 & !q8 & !q7 & !q6
        & !q5 & !q4 & q3 & q2 & !q1 & !q0
# !q15 & !q14 & !q13 & !q12 & !q11 & !q10 & !q9 & !q8 & !q7 & !q6
        & !q5 & !q4 & q3 & q2 & q1 & !q0 ;

q0 := !q15 & !q14 & !q13 & !q12 & !q11 & !q10 & !q9 & !q8 & !q7 & !q6
        & !q5 & !q4 & !q3 & !q2 & !q1 & !q0
# !q15 & !q14 & !q13 & !q12 & !q11 & !q10 & !q9 & !q8 & !q7 & !q6
        & !q5 & !q4 & !q3 & !q2 & !q1 & q0
# !q15 & !q14 & !q13 & !q12 & !q11 & !q10 & !q9 & !q8 & !q7 & !q6
        & !q5 & !q4 & !q3 & q2 & q1 & !q0
# !q15 & !q14 & !q13 & !q12 & !q11 & !q10 & !q9 & !q8 & !q7 & !q6
        & !q5 & !q4 & !q3 & q2 & q1 & q0 ;

end
```

4-Bit Gray Code Counter

```
module gray4 ;

title ' 4-bit gray code counter '

clk pin ;
rst pin ;
q3, q2, q1, q0 pin istype 'reg';

count = [q3..q0] ;

equations

count.ar = rst ;
count.clk = clk ;

 q3 := !q3 & q2 & !q1 & !q0
                    # q3 & q2 & !q1 & !q0
                    # q3 & q2 & !q1 & q0
                    # q3 & q2 & q1 & q0
                    # q3 & q2 & q1 & !q0
                    # q3 & !q2 & q1 & !q0
                    # q3 & !q2 & q1 & q0
                    # q3 & !q2 & !q1 & q0 ;

q2 := !q3 & !q2 & q1 & !q0
                    # !q3 & q2 & q1 & !q0
                    # !q3 & q2 & q1 & q0
                    # !q3 & q2 & !q1 & q0
                    # !q3 & q2 & !q1 & !q0
                    # q3 & q2 & !q1 & !q0
                    # q3 & q2 & !q1 & q0
                    # q3 & q2 & q1 & q0 ;

q1 := !q3 & !q2 & !q1 & q0
                    # !q3 & !q2 & q1 & q0
                    # !q3 & !q2 & q1 & !q0
                    # !q3 & q2 & q1 & !q0
                    # q3 & q2 & !q1 & q0
                    # q3 & q2 & q1 & q0
                    # q3 & q2 & q1 & !q0
                    # q3 & !q2 & q1 & !q0 ;

q0 := !q3 & !q2 & !q1 & !q0
                    # !q3 & !q2 & !q1 & q0
                    # !q3 & q2 & q1 & !q0
                    # !q3 & q2 & q1 & q0
```

```
                # q3 & q2 & !q1 & !q0
                # q3 & q2 & !q1 & q0
                # q3 & !q2 & q1 & !q0
                # q3 & !q2 & q1 & q0 ;

end gray4 ;
```

Timer/Counter

```
module p2

" Timer/Counter
" Two 8-bit loadable registers, mux, counter, and comparator
" This code is simular to Prep2

"Inputs
      CLK,RST,LDPRE,LDCOMP,SEL                pin ;
      DA7,DA6,DA5,DA4,DA3,DA2,DA1,DA0         pin ;
      DB7,DB6,DB5,DB4,DB3,DB2,DB1,DB0         pin ;
"Outputs
      O7,O6,O5,O4,O3,O2,O1,O0                 pin istype 'buffer,reg_t';
      LOAD0,LOAD1,LOAD2,LOAD3                 node istype 'buffer';
      LA7,LA6,LA5,LA4,LA3,LA2,LA1,LA0         node istype 'buffer,reg_d';
      LB7,LB6,LB5,LB4,LB3,LB2,LB1,LB0         node istype 'buffer,reg_d';

      LA  = [LA7,LA6,LA5,LA4,LA3,LA2,LA1,LA0];
      LB  = [LB7,LB6,LB5,LB4,LB3,LB2,LB1,LB0];
      DA  = [DA7,DA6,DA5,DA4,DA3,DA2,DA1,DA0];
      DB  = [DB7,DB6,DB5,DB4,DB3,DB2,DB1,DB0];
      OUT = [O7,O6,O5,O4,O3,O2,O1,O0];

      LOAD = LOAD0 # LOAD1 # LOAD2 # LOAD3;
      LOADB = !LOAD0 & !LOAD1 & !LOAD2 & !LOAD3;

      H,L,C,X = 1,0,.C.,.X.;

Equations

      LOAD0 = !O0.Q & LB0.Q
            # O0.Q & !LB0.Q
            # !O1.Q & LB1.Q
            # O1.Q & !LB1.Q;
      LOAD1 = !O2.Q & LB2.Q
            # O2.Q & !LB2.Q
            # !O3.Q & LB3.Q
            # O3.Q & !LB3.Q;
      LOAD2 = !O4.Q & LB4.Q
            # O4.Q & !LB4.Q
            # !O5.Q & LB5.Q
            # O5.Q & !LB5.Q;
      LOAD3 = !O6.Q & LB6.Q
            # O6.Q & !LB6.Q
            # !O7.Q & LB7.Q
            # O7.Q & !LB7.Q;
```

```
LA.CLK = CLK;
LB.CLK = CLK;
OUT.CLK = CLK;
LA.AR = RST;
LB.AR = RST;
OUT.AR = RST;

LA.D = DB & LDPRE # LA.Q & !LDPRE;          "INPUT LATCH
LB.D = DB & LDCOMP # LB.Q & !LDCOMP;        "INPUT LATCH

O0.T = LOAD                                 "COUNT
   #  LOADB & !SEL & (DA0 $ O0.Q)           "LOAD DATA FROM PIN
   #  LOADB &  SEL & (LA0.Q $ O0.Q);        "LOAD LATCHED DATA

O1.T = LOAD & O0.Q
   #  LOADB & !SEL & (DA1 $ O1.Q)
   #  LOADB &  SEL & (LA1.Q $ O1.Q);

O2.T = LOAD & O0.Q & O1.Q
   #  LOADB & !SEL & (DA2 $ O2.Q)
   #  LOADB &  SEL & (LA2.Q $ O2.Q);

O3.T = LOAD & O0.Q & O1.Q & O2.Q
   #  LOADB & !SEL & (DA3 $ O3.Q)
   #  LOADB &  SEL & (LA3.Q $ O3.Q);

O4.T = LOAD & O0.Q & O1.Q & O2.Q & O3.Q
   #  LOADB & !SEL & (DA4 $ O4.Q)
   #  LOADB &  SEL & (LA4.Q $ O4.Q);

O5.T = LOAD & O0.Q & O1.Q & O2.Q & O3.Q & O4.Q
   #  LOADB & !SEL & (DA5 $ O5.Q)
   #  LOADB &  SEL & (LA5.Q $ O5.Q);

O6.T = LOAD & O0.Q & O1.Q & O2.Q & O3.Q & O4.Q & O5.Q
   #  LOADB & !SEL & (DA6 $ O6.Q)
   #  LOADB &  SEL & (LA6.Q $ O6.Q);

O7.T = LOAD & O0.Q & O1.Q & O2.Q & O3.Q & O4.Q & O5.Q & O6.Q
   #  LOADB & !SEL & (DA7 $ O7.Q)
   #  LOADB &  SEL & (LA7.Q $ O7.Q);


end
```

16-Bit Loadable Binary Counter - 2 reps

```
module p7

" 16-bit loadable binary counter - 2 reps
" This is simular to Prep 7

"Inputs
      CLK,RST,LOAD,CE                  pin ;
      D15,D14,D13,D12,D11,D10,D9,D8    pin ;
      D7,D6,D5,D4,D3,D2,D1,D0          pin ;
"Outputs
      Q15,Q14,Q13,Q12,Q11,Q10,Q9,Q8    node istype 'buffer,reg_t';
      Q7,Q6,Q5,Q4,Q3,Q2,Q1,Q0          node istype 'buffer,reg_t';
      O15,O14,O13,O12,O11,O10,O9,O8    pin istype 'buffer,reg_t';
      O7,O6,O5,O4,O3,O2,O1,O0          pin istype 'buffer,reg_t';

        DIN  = [D15,D14,D13,D12,D11,D10,D9,D8,D7,D6,D5,D4,D3,D2,D1,D0];
      QOUT = [Q15,Q14,Q13,Q12,Q11,Q10,Q9,Q8,Q7,Q6,Q5,Q4,Q3,Q2,Q1,Q0];
      OOUT = [O15,O14,O13,O12,O11,O10,O9,O8,O7,O6,O5,O4,O3,O2,O1,O0];

         C,X = .C.,.X.;

Equations

      QOUT.CLK = CLK;
      QOUT.AR  = RST;
      OOUT.CLK = CLK;
      OOUT.AR  = RST;


      Q15.T = CE & !LOAD & Q14.Q & Q13.Q & Q12.Q & Q11.Q & Q10.Q & Q9.Q &
Q8.Q
            & Q7.Q & Q6.Q & Q5.Q & Q4.Q & Q3.Q & Q2.Q & Q1.Q & Q0.Q
            # LOAD & (Q15.Q $ D15);
      Q14.T = CE & !LOAD & Q13.Q & Q12.Q & Q11.Q & Q10.Q & Q9.Q & Q8.Q
            & Q7.Q & Q6.Q & Q5.Q & Q4.Q & Q3.Q & Q2.Q & Q1.Q & Q0.Q
            # LOAD & (Q14.Q $ D14);
      Q13.T = CE & !LOAD & Q12.Q & Q11.Q & Q10.Q & Q9.Q & Q8.Q
            & Q7.Q & Q6.Q & Q5.Q & Q4.Q & Q3.Q & Q2.Q & Q1.Q & Q0.Q
            # LOAD & (Q13.Q $ D13);
      Q12.T = CE & !LOAD & Q11.Q & Q10.Q & Q9.Q & Q8.Q
            & Q7.Q & Q6.Q & Q5.Q & Q4.Q & Q3.Q & Q2.Q & Q1.Q & Q0.Q
            # LOAD & (Q12.Q $ D12);
      Q11.T = CE & !LOAD & Q10.Q & Q9.Q & Q8.Q
            & Q7.Q & Q6.Q & Q5.Q & Q4.Q & Q3.Q & Q2.Q & Q1.Q & Q0.Q
            # LOAD & (Q11.Q $ D11);
```

```
    Q10.T = CE & !LOAD & Q9.Q & Q8.Q
        & Q7.Q & Q6.Q & Q5.Q & Q4.Q & Q3.Q & Q2.Q & Q1.Q & Q0.Q
        # LOAD & (Q10.Q $ D10);
     Q9.T = CE & !LOAD & Q8.Q & Q7.Q & Q6.Q & Q5.Q & Q4.Q & Q3.Q & Q2.Q
& Q1.Q & Q0.Q
        # LOAD & (Q9.Q $ D9);
     Q8.T = CE & !LOAD & Q7.Q & Q6.Q & Q5.Q & Q4.Q & Q3.Q & Q2.Q & Q1.Q
& Q0.Q
        # LOAD & (Q8.Q $ D8);
    Q7.T = CE & !LOAD & Q6.Q & Q5.Q & Q4.Q & Q3.Q & Q2.Q & Q1.Q & Q0.Q
        # LOAD & (Q7.Q $ D7);
    Q6.T = CE & !LOAD & Q5.Q & Q4.Q & Q3.Q & Q2.Q & Q1.Q & Q0.Q
        # LOAD & (Q6.Q $ D6);
    Q5.T = CE & !LOAD & Q4.Q & Q3.Q & Q2.Q & Q1.Q & Q0.Q
        # LOAD & (Q5.Q $ D5);
    Q4.T = CE & !LOAD & Q3.Q & Q2.Q & Q1.Q & Q0.Q
        # LOAD & (Q4.Q $ D4);
    Q3.T = CE & !LOAD & Q2.Q & Q1.Q & Q0.Q
        # LOAD & (Q3.Q $ D3);
    Q2.T = CE & !LOAD & Q1.Q & Q0.Q
        # LOAD & (Q2.Q $ D2);
    Q1.T = CE & !LOAD & Q0.Q
        # LOAD & (Q1.Q $ D1);
    Q0.T = CE & !LOAD
        # LOAD & (Q0.Q $ D0);

    O15.T = CE & !LOAD & O14.Q & O13.Q & O12.Q & O11.Q & O10.Q & O9.Q &
O8.Q
        & O7.Q & O6.Q & O5.Q & O4.Q & O3.Q & O2.Q & O1.Q & O0.Q
        # LOAD & (O15.Q $ Q15.Q);
    O14.T = CE & !LOAD & O13.Q & O12.Q & O11.Q & O10.Q & O9.Q & O8.Q
        & O7.Q & O6.Q & O5.Q & O4.Q & O3.Q & O2.Q & O1.Q & O0.Q
        # LOAD & (O14.Q $ Q14.Q);
    O13.T = CE & !LOAD & O12.Q & O11.Q & O10.Q & O9.Q & O8.Q
        & O7.Q & O6.Q & O5.Q & O4.Q & O3.Q & O2.Q & O1.Q & O0.Q
        # LOAD & (O13.Q $ Q13.Q);
    O12.T = CE & !LOAD & O11.Q & O10.Q & O9.Q & O8.Q
        & O7.Q & O6.Q & O5.Q & O4.Q & O3.Q & O2.Q & O1.Q & O0.Q
        # LOAD & (O12.Q $ Q12.Q);
    O11.T = CE & !LOAD & O10.Q & O9.Q & O8.Q
        & O7.Q & O6.Q & O5.Q & O4.Q & O3.Q & O2.Q & O1.Q & O0.Q
        # LOAD & (O11.Q $ Q11.Q);
    O10.T = CE & !LOAD & O9.Q & O8.Q
        & O7.Q & O6.Q & O5.Q & O4.Q & O3.Q & O2.Q & O1.Q & O0.Q
        # LOAD & (O10.Q $ Q10.Q);
```

```
        O9.T = CE & !LOAD & O8.Q & O7.Q & O6.Q & O5.Q & O4.Q & O3.Q & O2.Q
& O1.Q & O0.Q
            # LOAD & (O9.Q $ Q9.Q);
        O8.T = CE & !LOAD & O7.Q & O6.Q & O5.Q & O4.Q & O3.Q & O2.Q & O1.Q
& O0.Q
            # LOAD & (O8.Q $ Q8.Q);
        O7.T = CE & !LOAD & O6.Q & O5.Q & O4.Q & O3.Q & O2.Q & O1.Q & O0.Q
            # LOAD & (O7.Q $ Q7.Q);
        O6.T = CE & !LOAD & O5.Q & O4.Q & O3.Q & O2.Q & O1.Q & O0.Q
            # LOAD & (O6.Q $ Q6.Q);
        O5.T = CE & !LOAD & O4.Q & O3.Q & O2.Q & O1.Q & O0.Q
            # LOAD & (O5.Q $ Q5.Q);
        O4.T = CE & !LOAD & O3.Q & O2.Q & O1.Q & O0.Q
            # LOAD & (O4.Q $ Q4.Q);
        O3.T = CE & !LOAD & O2.Q & O1.Q & O0.Q
            # LOAD & (O3.Q $ Q3.Q);
        O2.T = CE & !LOAD & O1.Q & O0.Q
            # LOAD & (O2.Q $ Q2.Q);
        O1.T = CE & !LOAD & O0.Q
            # LOAD & (O1.Q $ Q1.Q);
        O0.T = CE & !LOAD
            # LOAD & (O0.Q $ Q0.Q);

end
```

16-Bit Synchronous Prescaled Counter - 2 reps

```
module p8

"16-bit Synchronous prescaled counter - 2 reps
"This circuit can divede the clock frequency by any value up to 65536
"This is simular to Prep 8

"Inputs
      CLK,RST,LOAD,CE                  pin ;
      D15,D14,D13,D12,D11,D10,D9,D8    pin ;
      D7,D6,D5,D4,D3,D2,D1,D0          pin ;
"Outputs
      Q15,Q14,Q13,Q12,Q11,Q10,Q9,Q8    node istype 'buffer,reg_t';
      Q7,Q6,Q5,Q4,Q3,Q2,Q1,Q0          node istype 'buffer,reg_t';
      O15,O14,O13,O12,O11,O10,O9,O8    pin istype 'buffer,reg_t';
      O7,O6,O5,O4,O3,O2,O1,O0          pin istype 'buffer,reg_t';

        DIN  = [D15,D14,D13,D12,D11,D10,D9,D8,D7,D6,D5,D4,D3,D2,D1,D0];
      QOUT = [Q15,Q14,Q13,Q12,Q11,Q10,Q9,Q8,Q7,Q6,Q5,Q4,Q3,Q2,Q1,Q0];
      OOUT = [O15,O14,O13,O12,O11,O10,O9,O8,O7,O6,O5,O4,O3,O2,O1,O0];

         C,X = .C.,.X.;

Equations

      QOUT.CLK = CLK;
      QOUT.AR  = RST;
      OOUT.CLK = CLK;
      OOUT.AR  = RST;


      Q15.T = CE & !LOAD & Q14.Q & Q13.Q & Q12.Q & Q11.Q & Q10.Q & Q9.Q &
Q8.Q
            & Q7.Q & Q6.Q & Q5.Q & Q4.Q & Q3.Q & Q2.Q & Q1.Q & Q0.Q
            # LOAD & (Q15.Q $ D15);
      Q14.T = CE & !LOAD & Q13.Q & Q12.Q & Q11.Q & Q10.Q & Q9.Q & Q8.Q
            & Q7.Q & Q6.Q & Q5.Q & Q4.Q & Q3.Q & Q2.Q & Q1.Q & Q0.Q
            # LOAD & (Q14.Q $ D14);
      Q13.T = CE & !LOAD & Q12.Q & Q11.Q & Q10.Q & Q9.Q & Q8.Q
            & Q7.Q & Q6.Q & Q5.Q & Q4.Q & Q3.Q & Q2.Q & Q1.Q & Q0.Q
            # LOAD & (Q13.Q $ D13);
      Q12.T = CE & !LOAD & Q11.Q & Q10.Q & Q9.Q & Q8.Q
            & Q7.Q & Q6.Q & Q5.Q & Q4.Q & Q3.Q & Q2.Q & Q1.Q & Q0.Q
            # LOAD & (Q12.Q $ D12);
      Q11.T = CE & !LOAD & Q10.Q & Q9.Q & Q8.Q
```

```
        & Q7.Q & Q6.Q & Q5.Q & Q4.Q & Q3.Q & Q2.Q & Q1.Q & Q0.Q
        # LOAD & (Q11.Q $ D11);
    Q10.T = CE & !LOAD & Q9.Q & Q8.Q
        & Q7.Q & Q6.Q & Q5.Q & Q4.Q & Q3.Q & Q2.Q & Q1.Q & Q0.Q
        # LOAD & (Q10.Q $ D10);
    Q9.T = CE & !LOAD & Q8.Q & Q7.Q & Q6.Q & Q5.Q & Q4.Q & Q3.Q & Q2.Q
& Q1.Q & Q0.Q
        # LOAD & (Q9.Q $ D9);
    Q8.T = CE & !LOAD & Q7.Q & Q6.Q & Q5.Q & Q4.Q & Q3.Q & Q2.Q & Q1.Q
& Q0.Q
        # LOAD & (Q8.Q $ D8);
    Q7.T = CE & !LOAD & Q6.Q & Q5.Q & Q4.Q & Q3.Q & Q2.Q & Q1.Q & Q0.Q
        # LOAD & (Q7.Q $ D7);
    Q6.T = CE & !LOAD & Q5.Q & Q4.Q & Q3.Q & Q2.Q & Q1.Q & Q0.Q
        # LOAD & (Q6.Q $ D6);
    Q5.T = CE & !LOAD & Q4.Q & Q3.Q & Q2.Q & Q1.Q & Q0.Q
        # LOAD & (Q5.Q $ D5);
    Q4.T = CE & !LOAD & Q3.Q & Q2.Q & Q1.Q & Q0.Q
        # LOAD & (Q4.Q $ D4);
    Q3.T = CE & !LOAD & Q2.Q & Q1.Q & Q0.Q
        # LOAD & (Q3.Q $ D3);
    Q2.T = CE & !LOAD & Q1.Q & Q0.Q
        # LOAD & (Q2.Q $ D2);
    Q1.T = CE & !LOAD & Q0.Q
        # LOAD & (Q1.Q $ D1);
    Q0.T = CE & !LOAD
        # LOAD & (Q0.Q $ D0);

    O15.T = CE & !LOAD & O14.Q & O13.Q & O12.Q & O11.Q & O10.Q & O9.Q &
O8.Q
        & O7.Q & O6.Q & O5.Q & O4.Q & O3.Q & O2.Q & O1.Q & O0.Q
        # LOAD & (O15.Q $ Q15.Q);
    O14.T = CE & !LOAD & O13.Q & O12.Q & O11.Q & O10.Q & O9.Q & O8.Q
        & O7.Q & O6.Q & O5.Q & O4.Q & O3.Q & O2.Q & O1.Q & O0.Q
        # LOAD & (O14.Q $ Q14.Q);
    O13.T = CE & !LOAD & O12.Q & O11.Q & O10.Q & O9.Q & O8.Q
        & O7.Q & O6.Q & O5.Q & O4.Q & O3.Q & O2.Q & O1.Q & O0.Q
        # LOAD & (O13.Q $ Q13.Q);
    O12.T = CE & !LOAD & O11.Q & O10.Q & O9.Q & O8.Q
        & O7.Q & O6.Q & O5.Q & O4.Q & O3.Q & O2.Q & O1.Q & O0.Q
        # LOAD & (O12.Q $ Q12.Q);
    O11.T = CE & !LOAD & O10.Q & O9.Q & O8.Q
        & O7.Q & O6.Q & O5.Q & O4.Q & O3.Q & O2.Q & O1.Q & O0.Q
        # LOAD & (O11.Q $ Q11.Q);
    O10.T = CE & !LOAD & O9.Q & O8.Q
```

```
        & O7.Q & O6.Q & O5.Q & O4.Q & O3.Q & O2.Q & O1.Q & O0.Q
        # LOAD & (O10.Q $ Q10.Q);
    O9.T = CE & !LOAD & O8.Q & O7.Q & O6.Q & O5.Q & O4.Q & O3.Q & O2.Q
& O1.Q & O0.Q
        # LOAD & (O9.Q $ Q9.Q);
    O8.T = CE & !LOAD & O7.Q & O6.Q & O5.Q & O4.Q & O3.Q & O2.Q & O1.Q
& O0.Q
        # LOAD & (O8.Q $ Q8.Q);
    O7.T = CE & !LOAD & O6.Q & O5.Q & O4.Q & O3.Q & O2.Q & O1.Q & O0.Q
        # LOAD & (O7.Q $ Q7.Q);
    O6.T = CE & !LOAD & O5.Q & O4.Q & O3.Q & O2.Q & O1.Q & O0.Q
        # LOAD & (O6.Q $ Q6.Q);
    O5.T = CE & !LOAD & O4.Q & O3.Q & O2.Q & O1.Q & O0.Q
        # LOAD & (O5.Q $ Q5.Q);
    O4.T = CE & !LOAD & O3.Q & O2.Q & O1.Q & O0.Q
        # LOAD & (O4.Q $ Q4.Q);
    O3.T = CE & !LOAD & O2.Q & O1.Q & O0.Q
        # LOAD & (O3.Q $ Q3.Q);
    O2.T = CE & !LOAD & O1.Q & O0.Q
        # LOAD & (O2.Q $ Q2.Q);
    O1.T = CE & !LOAD & O0.Q
        # LOAD & (O1.Q $ Q1.Q);
    O0.T = CE & !LOAD
        # LOAD & (O0.Q $ Q0.Q);


end
```

Serial CRC Generator using a 16-Bit LFSR g(x) = x16 + x12 + x5 + 1

```
module crc8s

title ' serial crc generator using a 16-bit lfsr g(x) = x16 + x12 + x5 + 1'

clk pin ;
set pin ;
din pin ;
x15..x0 pin istype 'reg,buffer,xor' ;

crc_sum = [x15..x0] ;

equations

x0 := din $ x15 ;
x1 := x0 ;
x2 := x1 ;
x3 := x2 ;
x4 := x3 ;
x5 := x4 & din & x15 ;
x6 := x5 ;
x7 := x6 ;
x8 := x7 ;
x9 := x8 ;
x10 := x9 ;
x11 := x10 ;
x12 := x11 & din & x15 ;
x13 := x12 ;
x14 := x13 ;
x15 := x14 ;

crc_sum.clk = clk ;
crc_sum.ap = set ;


end
```

3 to 8 Decoder

```
MODULE dec38

TITLE '3 to 8 decoder'

i2..i0 pin ;
o7..o0 pin ;

i = [i2..i0] ;
o = [o7..o0] ;

equations

o7 = i2 & i1 & i0 ;
o6 = i2 & i1 & !i0 ;
o5 = i2 & !i1 & i0 ;
o4 = i2 & !i1 & !i0 ;
o3 = !i2 & i1 & i0 ;
o2 = !i2 & i1 & !i0 ;
o1 = !i2 & !i1 & i0 ;
o0 = !i2 & !i1 & !i0 ;

end
```

8-Bit Loadable Data Register

```
MODULE dr8

TITLE '8-bit loadable data register'

d7..d0 pin ;
q7..q0 pin istype 'reg';

le pin ;
clk pin ;
rst pin ;

d = [d7..d0] ;
q = [q7..q0] ;

equations

q :=      le & d
          #!le & q ;
q.ar = rst ;
q.clk = clk ;


END
```

DRAM Controller

```
Module  dram_ctl
Title   'This is a DRAM controller for a R3081E Microprocessor. This DRAM
        Controller can be used to access 16 MBytes of DRAM" This controller
        is synchronized to the R3081E to guarantee fast operation. This
        DRAM controller uses the R3081E control signals as inputs and generated
        synchrozied acknowledgments back to the R3081E to terminate the DRAM
        access cycle. Signals to control for data buffers and the multiplexing
        of the Row and Column addresses and also provided'


MPCLK, MPRDn, MPWRn, MPRSTn, MPBURSTn                   pin;
BE3n, BE2n, BE1n, BE0n                                  pin;
DRAMCSn, STARTCYCLE, RFTIMEn, OEn                       pin;
DRSMST4, DRSMST3, DRSMST2, DRSMST1, DRSMST0             pin  istype 'reg';
DRRASn                                                  pin  istype 'reg';
DRCAS3n, DRCAS2n, DRCAS1n, DRCAS0n                      pin  istype 'reg';
DRRDn, DRWRn, DRRDLE, DRRWCLn, DRRDCEn, DRACKn          pin  istype 'reg';
RFSTARTn                                                pin  istype 'reg';


dst = [DRSMST4, DRSMST3, DRSMST2, DRSMST1, DRSMST0];

dst0    = ^h00;         "Refresh
dst1    = ^h10;         "Refresh
dst2    = ^h12;         "Refresh
dst3    = ^h16;         "Refresh
dst4    = ^h14;         "Refresh
dst5    = ^h04;         "Refresh
dst6    = ^h0c;         "Idle
dst7    = ^h1d;         "Read
dst8    = ^h1c;         "Read
dst9    = ^h1e;         "Read
dst10   = ^h0e;         "Read
dst11   = ^h0f;         "Burst Read
dst12   = ^h1f;         "Burst Read
dst13   = ^h17;         "Burst Read
dst14   = ^h15;         "Burst Read
dst15   = ^h11;         "Burst Read
dst16   = ^h13;         "Burst Read
dst17   = ^h03;         "Burst Read
dst18   = ^h02;         "Burst Read
dst19   = ^h06;         "Burst Read
dst20   = ^h08;         "Write
dst21   = ^h09;         "Write
dst22   = ^h0d;         "Write
"dst23  = ^h01;          unused state
```

```
"dst24   = ^h05;          unused state
"dst25   = ^h07;          unused state
"dst26   = ^h0a;          unused state
"dst27   = ^h0b;          unused state
"dst28   = ^h18;          unused state
"dst29   = ^h19;          unused state
"dst30   = ^h1a;          unused state
"dst31   = ^h1b;          unused state


DRCASn = [DRCAS3n, DRCAS2n, DRCAS1n, DRCAS0n];
DROUTn = [DRRDn, DRWRn, DRRDLE, DRRWCLn, DRRDCEn, DRACKn, RFSTARTn];

        x = .x.;  "Don't Care
        z = .z.;  "High Impedance
        c = .c.;  "Clock
        u = .u.;  "Undefined

equations

dst.C           = !MPCLK;
dst.OE          = !OEn;
dst.AP          = 0;
dst.AR          = 0;

DRRASn.C        = !MPCLK;
DRRASn.OE       = !OEn;
DRRASn.AP       = 0;
DRRASn.AR       = 0;

DRCASn.C        = MPCLK;
DRCASn.OE       = !OEn;
DRCASn.AP       = 0;
DRCASn.AR       = 0;

DROUTn.C        = !MPCLK;
DROUTn.OE       = !OEn;
DROUTn.AP       = 0;
DROUTn.AR       = 0;


!DRRASn         := (((dst == dst2)  &  MPRSTn)
                    # ((dst == dst3)  &  MPRSTn)
                    # ((dst == dst6)  &  RFTIMEn & STARTCYCLE & !MPRDn &
MPWRn & MPRSTn)
                    # ((dst == dst6)  &  RFTIMEn & STARTCYCLE &  MPRDn &
!MPWRn & MPRSTn)
                    # ((dst == dst7)  & !DRAMCSn & !MPRDn &  MPWRn & MPRSTn)
```

```
                # ((dst == dst8)  & !DRAMCSn & !MPRDn &  MPWRn & MPRSTn)
                # ((dst == dst9)  & !DRAMCSn & !MPRDn &  MPWRn & MPRSTn)
                # ((dst == dst10) & !DRAMCSn & !MPRDn &  MPWRn & MPRSTn &
!MPBURSTn)
                # ((dst == dst11) & !DRAMCSn & !MPRDn &  MPWRn & MPRSTn)
                # ((dst == dst12) & !DRAMCSn & !MPRDn &  MPWRn & MPRSTn)
                # ((dst == dst13) & !DRAMCSn & !MPRDn &  MPWRn & MPRSTn)
                # ((dst == dst14) & !DRAMCSn & !MPRDn &  MPWRn & MPRSTn)
                # ((dst == dst15) & !DRAMCSn & !MPRDn &  MPWRn & MPRSTn)
                # ((dst == dst16) & !DRAMCSn & !MPRDn &  MPWRn & MPRSTn)
                # ((dst == dst17) & !DRAMCSn & !MPRDn &  MPWRn & MPRSTn)
                # ((dst == dst18) & !DRAMCSn & !MPRDn &  MPWRn & MPRSTn)
                # ((dst == dst20) & !DRAMCSn &  MPRDn & !MPWRn & MPRSTn)
                # ((dst == dst21) & !DRAMCSn &  MPRDn & !MPWRn & MPRSTn));

!DRCAS3n     := (((dst == dst2)  &  MPRSTn)
                # ((dst == dst3)  &  MPRSTn)
                 # ((dst == dst8)  & !DRAMCSn & !BE3n  & !MPRDn &  MPWRn &
MPRSTn)
                 # ((dst == dst9)  & !DRAMCSn & !BE3n  & !MPRDn &  MPWRn &
MPRSTn)
                 # ((dst == dst11) & !DRAMCSn & !BE3n  & !MPRDn &  MPWRn &
MPRSTn)
                 # ((dst == dst12) & !DRAMCSn & !BE3n  & !MPRDn &  MPWRn &
MPRSTn)
                 # ((dst == dst14) & !DRAMCSn & !BE3n  & !MPRDn &  MPWRn &
MPRSTn)
                 # ((dst == dst15) & !DRAMCSn & !BE3n  & !MPRDn &  MPWRn &
MPRSTn)
                 # ((dst == dst17) & !DRAMCSn & !BE3n  & !MPRDn &  MPWRn &
MPRSTn)
                 # ((dst == dst18) & !DRAMCSn & !BE3n  & !MPRDn &  MPWRn &
MPRSTn)
                 # ((dst == dst21) & !DRAMCSn & !BE3n  &  MPRDn & !MPWRn &
MPRSTn)
                 # ((dst == dst22) & !DRAMCSn & !BE3n  &  MPRDn & !MPWRn &
MPRSTn));

!DRCAS2n     := (((dst == dst2)  &  MPRSTn)
                # ((dst == dst3)  &  MPRSTn)
                 # ((dst == dst8)  & !DRAMCSn & !BE2n  & !MPRDn &  MPWRn &
MPRSTn)
                 # ((dst == dst9)  & !DRAMCSn & !BE2n  & !MPRDn &  MPWRn &
MPRSTn)
```

```
                # ((dst == dst11) & !DRAMCSn & !BE2n  & !MPRDn &  MPWRn &
MPRSTn)
                # ((dst == dst12) & !DRAMCSn & !BE2n  & !MPRDn &  MPWRn &
MPRSTn)
                # ((dst == dst14) & !DRAMCSn & !BE2n  & !MPRDn &  MPWRn &
MPRSTn)
                # ((dst == dst15) & !DRAMCSn & !BE2n  & !MPRDn &  MPWRn &
MPRSTn)
                # ((dst == dst17) & !DRAMCSn & !BE2n  & !MPRDn &  MPWRn &
MPRSTn)
                # ((dst == dst18) & !DRAMCSn & !BE2n  & !MPRDn &  MPWRn &
MPRSTn)
                # ((dst == dst21) & !DRAMCSn & !BE2n  &  MPRDn & !MPWRn &
MPRSTn)
                # ((dst == dst22) & !DRAMCSn & !BE2n  &  MPRDn & !MPWRn &
MPRSTn));

!DRCAS1n    := (((dst == dst2)  &  MPRSTn)
                # ((dst == dst3)  &  MPRSTn)
                # ((dst == dst8)  & !DRAMCSn & !BE1n  & !MPRDn &  MPWRn &
MPRSTn)
                # ((dst == dst9)  & !DRAMCSn & !BE1n  & !MPRDn &  MPWRn &
MPRSTn)
                # ((dst == dst11) & !DRAMCSn & !BE1n  & !MPRDn &  MPWRn &
MPRSTn)
                # ((dst == dst12) & !DRAMCSn & !BE1n  & !MPRDn &  MPWRn &
MPRSTn)
                # ((dst == dst14) & !DRAMCSn & !BE1n  & !MPRDn &  MPWRn &
MPRSTn)
                # ((dst == dst15) & !DRAMCSn & !BE1n  & !MPRDn &  MPWRn &
MPRSTn)
                # ((dst == dst17) & !DRAMCSn & !BE1n  & !MPRDn &  MPWRn &
MPRSTn)
                # ((dst == dst18) & !DRAMCSn & !BE1n  & !MPRDn &  MPWRn &
MPRSTn)
                # ((dst == dst21) & !DRAMCSn & !BE1n  &  MPRDn & !MPWRn &
MPRSTn)
                # ((dst == dst22) & !DRAMCSn & !BE1n  &  MPRDn & !MPWRn &
MPRSTn));

!DRCAS0n    := (((dst == dst2)  &  MPRSTn)
                # ((dst == dst3)  &  MPRSTn)
                # ((dst == dst8)  & !DRAMCSn & !BE0n  & !MPRDn &  MPWRn &
MPRSTn)
```

```
                # ((dst == dst9)  & !DRAMCSn & !BE0n  & !MPRDn &  MPWRn &
MPRSTn)
                # ((dst == dst11) & !DRAMCSn & !BE0n  & !MPRDn &  MPWRn &
MPRSTn)
                # ((dst == dst12) & !DRAMCSn & !BE0n  & !MPRDn &  MPWRn &
MPRSTn)
                # ((dst == dst14) & !DRAMCSn & !BE0n  & !MPRDn &  MPWRn &
MPRSTn)
                # ((dst == dst15) & !DRAMCSn & !BE0n  & !MPRDn &  MPWRn &
MPRSTn)
                # ((dst == dst17) & !DRAMCSn & !BE0n  & !MPRDn &  MPWRn &
MPRSTn)
                # ((dst == dst18) & !DRAMCSn & !BE0n  & !MPRDn &  MPWRn &
MPRSTn)
                # ((dst == dst21) & !DRAMCSn & !BE0n  &  MPRDn & !MPWRn &
MPRSTn)
                # ((dst == dst22) & !DRAMCSn & !BE0n  &  MPRDn & !MPWRn &
MPRSTn));

!DRRDn      := (((dst == dst7)  & !DRAMCSn & !MPRDn &  MPWRn & MPRSTn)
                # ((dst == dst8)  & !DRAMCSn & !MPRDn &  MPWRn & MPRSTn)
                # ((dst == dst9)  & !DRAMCSn & !MPRDn &  MPWRn & MPRSTn)
                # ((dst == dst10) & !DRAMCSn & !MPRDn &  MPWRn & MPRSTn)
                # ((dst == dst11) & !DRAMCSn & !MPRDn &  MPWRn & MPRSTn &
!MPBURSTn)
                # ((dst == dst12) & !DRAMCSn & !MPRDn &  MPWRn & MPRSTn &
!MPBURSTn)
                # ((dst == dst13) & !DRAMCSn & !MPRDn &  MPWRn & MPRSTn &
!MPBURSTn)
                # ((dst == dst14) & !DRAMCSn & !MPRDn &  MPWRn & MPRSTn &
!MPBURSTn)
                # ((dst == dst15) & !DRAMCSn & !MPRDn &  MPWRn & MPRSTn &
!MPBURSTn)
                # ((dst == dst16) & !DRAMCSn & !MPRDn &  MPWRn & MPRSTn &
!MPBURSTn)
                # ((dst == dst17) & !DRAMCSn & !MPRDn &  MPWRn & MPRSTn &
!MPBURSTn)
                # ((dst == dst18) & !DRAMCSn & !MPRDn &  MPWRn & MPRSTn &
!MPBURSTn)
                # ((dst == dst19) & !DRAMCSn & !MPRDn &  MPWRn & MPRSTn &
!MPBURSTn));

!DRWRn      := (((dst == dst6)  & !DRAMCSn &  MPRDn & !MPWRn & MPRSTn &
RFTIMEn                                     &  STARTCYCLE)
                # ((dst == dst20) & !DRAMCSn &  MPRDn & !MPWRn & MPRSTn)
```

```
                  # ((dst == dst21) & !DRAMCSn &  MPRDn & !MPWRn & MPRSTn));

DRRDLE       := (((dst == dst8)  & !DRAMCSn & !MPRDn &  MPWRn & MPRSTn)
                  # ((dst == dst11) & !DRAMCSn & !MPRDn &  MPWRn & MPRSTn)
                  # ((dst == dst14) & !DRAMCSn & !MPRDn &  MPWRn & MPRSTn)
                  # ((dst == dst17) & !DRAMCSn & !MPRDn &  MPWRn & MPRSTn));

!DRRWCLn     := (((dst == dst7)  & !DRAMCSn & !MPRDn &  MPWRn & MPRSTn)
                  # ((dst == dst8)  & !DRAMCSn & !MPRDn &  MPWRn & MPRSTn)
                  # ((dst == dst9)  & !DRAMCSn & !MPRDn &  MPWRn & MPRSTn)
                  # ((dst == dst10) & !DRAMCSn & !MPRDn &  MPWRn & MPRSTn &
!MPBURSTn)
                  # ((dst == dst11) & !DRAMCSn & !MPRDn &  MPWRn & MPRSTn)
                  # ((dst == dst12) & !DRAMCSn & !MPRDn &  MPWRn & MPRSTn)
                  # ((dst == dst13) & !DRAMCSn & !MPRDn &  MPWRn & MPRSTn)
                  # ((dst == dst14) & !DRAMCSn & !MPRDn &  MPWRn & MPRSTn)
                  # ((dst == dst15) & !DRAMCSn & !MPRDn &  MPWRn & MPRSTn)
                  # ((dst == dst16) & !DRAMCSn & !MPRDn &  MPWRn & MPRSTn)
                  # ((dst == dst17) & !DRAMCSn & !MPRDn &  MPWRn & MPRSTn)
                  # ((dst == dst18) & !DRAMCSn & !MPRDn &  MPWRn & MPRSTn)
                  # ((dst == dst20) & !DRAMCSn &  MPRDn & !MPWRn & MPRSTn)
                  # ((dst == dst21) & !DRAMCSn &  MPRDn & !MPWRn & MPRSTn));


!DRRDCEn     := (((dst == dst9)  & !DRAMCSn & !MPRDn &  MPWRn & MPRSTn)
                  # ((dst == dst12) & !DRAMCSn & !MPRDn &  MPWRn & MPRSTn)
                  # ((dst == dst15) & !DRAMCSn & !MPRDn &  MPWRn & MPRSTn)
                  # ((dst == dst18) & !DRAMCSn & !MPRDn &  MPWRn & MPRSTn));

!DRACKn      := (((dst == dst9)  & !DRAMCSn & !MPRDn &  MPWRn & MPRSTn &
MPBURSTn)
                  # ((dst == dst15) & !DRAMCSn & !MPRDn &  MPWRn & MPRSTn)
                  # ((dst == dst20) & !DRAMCSn &  MPRDn & !MPWRn & MPRSTn));

!RFSTARTn    := (((dst == dst2)  & MPRSTn)
                  # ((dst == dst3)  & MPRSTn)
                  # ((dst == dst4)  & MPRSTn));


STATE_DIAGRAM[DRSMST4,DRSMST3,DRSMST2,DRSMST1,DRSMST0]

state dst0:       if (!MPRSTn) then dst0
                  else dst1;

state dst1:       goto dst2;
state dst2:       goto dst3;
```

```
state dst3:     goto dst4;
state dst4:     goto dst5;
state dst5:     goto dst6;

state dst6:     if (!RFTIMEn) then dst1
                else if (STARTCYCLE & !DRAMCSn & !MPRDn &  MPWRn & RFTIMEn) then dst7
                else if (STARTCYCLE & !DRAMCSn & MPRDn & !MPWRn & RFTIMEn) then dst20
                else dst6;

state dst7:     goto dst8;
state dst8:     goto dst9;
state dst9:     goto dst10;

state dst10:    if (!MPBURSTn) then dst11
                else dst6;

state dst11:    goto dst12;
state dst12:    goto dst13;
state dst13:    goto dst14;
state dst14:    goto dst15;
state dst15:    goto dst16;
state dst16:    goto dst17;
state dst17:    goto dst18;
state dst18:    goto dst19;
state dst19:    goto dst6;

state dst20:    goto dst21;
state dst21:    goto dst22;
state dst22:    goto dst6;

end
```

High Level Implementation of 16 to 8 Multiplexer

```
MODULE m16_8

" High level implementation of 16 to 8 multiplexer

a7..a0 pin ;
b7..b0 pin ;
y7..y0 pin ;
sel pin ;

a = [a7..a0] ;
b = [b7..b0] ;
y = [y7..y0] ;

equations

y =     sel & a
        # !sel & b ;


end
```

Dual 4 to 1 Mux

```
MODULE m41_2 ;

title ' Dual 4 to 1 mux'

ai3..ai0 pin ;
bi3..bi0 pin ;

as1, as0 pin ;
bs1, bs0 pin ;

ao pin ;
bo pin ;

ai = [ai3..ai0] ;
bi = [bi3..bi0] ;

equations

ao = !as1 & !as0 & ai0
          # !as1 & as0 & ai1
          # as1  & !as0 & ai2
          # as1 & as0 & ai3 ;

bo =      !bs1 & !bs0 & bi0
          # !bs1 & bs0 & bi1
          # bs1 & !bs0 & bi2
          # bs1 & bs0 & bi3 ;


END
```

Datapath Circuit - Two reps

```
module p1

" Datapath circuit - Two reps
" 4-1 8-bit mux feeds 8-bit data reg which feeds 8-bit shift register
" This is simular to Prep 1


"Inputs
      CLK,RST,S1,S0,OE_ID0,SL_ID15                pin ;
      IPD7,IPD6,IPD5,IPD4,IPD3,IPD2,IPD1,IPD0     pin ;
      ID7,ID6,ID5,ID4,ID3,ID2,ID1                 pin ;
      ID14,ID13,ID12,ID11,ID10,ID9,ID8            pin ;

"Outputs
      Q7,Q6,Q5,Q4,Q3,Q2,Q1,Q0       pin istype 'buffer,reg_d';    "last instance shift register
      R7,R6,R5,R4,R3,R2,R1,R0       node istype 'buffer,reg_d'; "last instance register

         B7,B6,B5,B4,B3,B2,B1,B0    node istype 'buffer,reg_d'; "first instance register
         C7,C6,C5,C4,C3,C2,C1,C0    node istype 'buffer,reg_d'; "first instance shift register

         B   = [B7,B6,B5,B4,B3,B2,B1,B0];
         C   = [C7,C6,C5,C4,C3,C2,C1,C0];
         SC  = [C6,C5,C4,C3,C2,C1,C0,C7];
      R   = [R7,R6,R5,R4,R3,R2,R1,R0];
      D0  = [IPD7,IPD6,IPD5,IPD4,IPD3,IPD2,IPD1,IPD0];
      D1  = [SL_ID15,ID14,ID13,ID12,ID11,ID10,ID9,ID8];
      D2  = [ID7,ID6,ID5,ID4,ID3,ID2,ID1,OE_ID0];
      D3  = [Q7,Q6,Q5,Q4,Q3,Q2,Q1,Q0];
      SF  = [Q6,Q5,Q4,Q3,Q2,Q1,Q0,Q7];

Equations
         B.CLK = CLK;
         C.CLK = CLK;
         R.CLK = CLK;
         D3.CLK = CLK;

         B.AR = RST;
         C.AR = RST;
         R.AR = RST;
         D3.AR = RST;

      B.D = !S1 & !S0 & D0
         # !S1 &  S0 & D1
         #  S1 & !S0 & D2
         #  S1 &  S0 & D3.PIN;
```

```
    C.D = !SL_ID15 & B.Q      "LOAD
        #  SL_ID15 & SC.Q;    "SHIFT

R.D = !S1 & !S0 & C.Q
    # !S1 &  S0 & D1
    #  S1 & !S0 & D2
    #  S1 &  S0 & D3.PIN;

D3.D = !SL_ID15 & R.Q     "LOAD
     #  SL_ID15 & SF.Q;   "SHIFT

D3.OE = OE_ID0;


end
```

Small State Machine - 8 inputs, 8 registered outputs

```
module p3

"Small state machine - 8 inputs, 8 registered outputs - 3 reps
"This is simular to Prep3

"Inputs
      CLK,RST,I7,I6,I5,I4,I3,I2,I1,I0   pin ;
"Outputs
      O7,O6,O5,O4,O3,O2,O1,O0          pin istype 'buffer,reg_d';
      B7,B6,B5,B4,B3,B2,B1,B0          node istype 'buffer,reg_d';
      C7,C6,C5,C4,C3,C2,C1,C0          node istype 'buffer,reg_d';
      Q,QB,QC                          node istype 'buffer,reg_d';

"State Register assignment
      OUT   = [B7,B6,B5,B4,B3,B2,B1,B0];
      inp     = [I7,I6,I5,I4,I3,I2,I1,I0];
      inp1    = [B7.Q,B6.Q,B5.Q,B4.Q,B3.Q,B2.Q,B1.Q,B0.Q];
      inp2    = [C7.Q,C6.Q,C5.Q,C4.Q,C3.Q,C2.Q,C1.Q,C0.Q];
      sreg    = [O7,O6,O5,O4,O3,O2,O1,O0, Q];
      sreg1   = [B7,B6,B5,B4,B3,B2,B1,B0,QB];
      sreg2   = [C7,C6,C5,C4,C3,C2,C1,C0,QC];
      START   = [ 0, 0, 0, 0, 0, 0, 0, 0, 0]; "00
      SA      = [ 1, 0, 0, 0, 0, 0, 1, 0, 0]; "82
      SA1     = [ 0, 0, 0, 0, 0, 1, 0, 0, 0]; "04
      SB      = [ 0, 0, 1, 0, 0, 0, 0, 0, 0]; "20
      SC      = [ 0, 1, 0, 0, 0, 0, 0, 0, 1]; "40
      SD      = [ 0, 0, 0, 0, 1, 0, 0, 0, 0]; "08
      SE      = [ 0, 0, 0, 1, 0, 0, 0, 1, 0]; "11
      SF      = [ 0, 0, 1, 1, 0, 0, 0, 0, 0]; "30
      SG      = [ 0, 0, 0, 0, 0, 0, 1, 0, 0]; "02
      SG1     = [ 1, 0, 0, 0, 0, 0, 0, 0, 0]; "80
      START1  = [ 0, 1, 0, 0, 0, 0, 0, 0, 0]; "40
      START2  = [ 0, 0, 0, 0, 0, 0, 0, 1, 0]; "01
      H,L,C,X = 1,0,.C.,.X.;

Equations
      sreg.ar     = !RST;
      sreg.clk    = CLK;
      sreg1.ar    = !RST;
      sreg1.clk   = CLK;
      sreg2.ar    = !RST;
      sreg2.clk   = CLK;

state_diagram sreg1       "first instance

state START:
```

```
        if (inp == ^h3C) then SA
        else START;

state START1:
        if (inp == ^h3C) then SA
        else START;

state START2:
        if (inp == ^h3C) then SA
        else START;

state SA:
        if (inp == ^h1F) then SB
        else if (inp == ^h2A) then SC
        else SA1;

state SA1:
        if (inp == ^h1F) then SB
        else if (inp == ^h2A) then SC
        else SA1;

state SB:
        if (inp == ^hAA) then SE
        else SF;

state SC:
        goto SD;

state SD:
        goto SG1;

state SE:
        goto START1;

state SF:
        goto SG;

state SG:
        goto START2;

state SG1:
        goto START2;

state_diagram sreg2       "second instance

state START:
        if (inp1 == ^h3C) then SA
        else START;
```

```
state START1:
        if (inp1 == ^h3C) then SA
        else START;

state START2:
        if (inp1 == ^h3C) then SA
        else START;

state SA:
        if (inp1 == ^h1F) then SB
        else if (inp1 == ^h2A) then SC
        else SA1;

state SA1:
        if (inp1 == ^h1F) then SB
        else if (inp1 == ^h2A) then SC
        else SA1;

state SB:
        if (inp1 == ^hAA) then SE
        else SF;

state SC:
        goto SD;

state SD:
        goto SG1;

state SE:
        goto START1;

state SF:
        goto SG;

state SG:
        goto START2;

state SG1:
        goto START2;

state_diagram sreg        "third instance

state START:
        if (inp2 == ^h3C) then SA
        else START;

state START1:
```

```
        if (inp2 == ^h3C) then SA
        else START;

state START2:
        if (inp2 == ^h3C) then SA
        else START;

state SA:
        if (inp2 == ^h1F) then SB
        else if (inp2 == ^h2A) then SC
        else SA1;

state SA1:
        if (inp2 == ^h1F) then SB
        else if (inp2 == ^h2A) then SC
        else SA1;

state SB:
        if (inp2 == ^hAA) then SE
        else SF;

state SC:
        goto SD;

state SD:
        goto SG1;

state SE:
        goto START1;

state SF:
        goto SG;

state SG:
        goto START2;

state SG1:
        goto START2;



end
```

Arithmetic Circuit: 4 x 4 multiplier, 8-bit adder, 8-bit register

```
module p5

"Arithmetic circuit: 4 x 4 multiplier, 8-bit adder, 8-bit register
"This is simular to Prep 5

"Inputs
      CLK,RST,MAC                   pin ;
      A3,A2,A1,A0,B3,B2,B1,B0       pin ;

"Outputs
      Q7,Q6,Q5,Q4,Q3,Q2,Q1,Q0        pin istype 'buffer,reg_d';

      C53,C52,C21,C22,C23,C24               node istype 'buffer';
      C31,C32,C33,C34,C41,C42,C43,C44       node istype 'buffer';
      S22,S23,S24,S32,S33,S34,S42,S43,S44   node istype 'buffer';

      C11 = (MAC & Q0.Q) & (A0 & B0);
      C12 = (MAC & Q1.Q) & (A1 & B0);
      C13 = (MAC & Q2.Q) & (A2 & B0);
      C14 = (MAC & Q3.Q) & (A3 & B0);

      S12 = (MAC & Q1.Q) $ (A1 & B0);
      S13 = (MAC & Q2.Q) $ (A2 & B0);
      S14 = (MAC & Q3.Q) $ (A3 & B0);



      C51 = S42 & C41;

      A  = [A3,A2,A1,A0];
      B  = [B3,B2,B1,B0];
                  Q  = [Q7,Q6,Q5,Q4,Q3,Q2,Q1,Q0];

      H,L,C,X = 1,0,.C.,.X.;

Equations

      C52 = S43 & C42 # S43 & C51 # C42 & C51;
      C53 = S44 & C43 # S44 & C52 # C43 & C52;

      C21 = S12 & C11 # S12 & (A0 & B1) # C11 & (A0 & B1);
      C22 = S13 & C12 # S13 & (A1 & B1) # C12 & (A1 & B1);
      C23 = S14 & C13 # S14 & (A2 & B1) # C13 & (A2 & B1);
      C24 = (MAC & Q4.Q) & C14 # (MAC & Q4.Q) & (A3 & B1) # C14 & (A3 & B1);

      S22 = S13 $ C12 $ (A1 & B1);
```

```
S23 = S14 $ C13 $ (A2 & B1);
S24 = (MAC & Q4.Q) $ C14 $ (A3 & B1);

C31 = S22 & C21 # S22 & (A0 & B2) # C21 & (A0 & B2);
C32 = S23 & C22 # S23 & (A1 & B2) # C22 & (A1 & B2);
C33 = S24 & C23 # S24 & (A2 & B2) # C23 & (A2 & B2);
C34 = (MAC & Q5.Q) & C24 # (MAC & Q5.Q) & (A3 & B2) # C24 & (A3 & B2);

S32 = S23 $ C22 $ (A1 & B2);
S33 = S24 $ C23 $ (A2 & B2);
S34 = (MAC & Q5.Q) $ C24 $ (A3 & B2);

C41 = S32 & C31 # S32 & (A0 & B3) # C31 & (A0 & B3);
C42 = S33 & C32 # S33 & (A1 & B3) # C32 & (A1 & B3);
C43 = S34 & C33 # S34 & (A2 & B3) # C33 & (A2 & B3);
C44 = (MAC & Q6.Q) & C34 # (MAC & Q6.Q) & (A3 & B3) # C34 & (A3 & B3);

S42 = S33 $ C32 $ (A1 & B3);
S43 = S34 $ C33 $ (A2 & B3);
S44 = (MAC & Q6.Q) $ C34 $ (A3 & B3);

Q.CLK = CLK;
Q.AR  = RST;

Q0.D = (MAC & Q0.Q) $ (A0 & B0);
Q1.D = S12 $ C11 $ (A0 & B1);
Q2.D = S22 $ C21 $ (A0 & B2);
Q3.D = S32 $ C31 $ (A0 & B3);
Q4.D = S42 $ C41;
Q5.D = S43 $ C42 $ (S42 & C41);
Q6.D = S44 $ C43 $ C52;
Q7.D = (MAC & Q7.Q) $ C44 $ C53;


end
```

1-Bit Accumulator

```
module p6

" 1--bit accumulator
" This is simular to Prep 6

"Inputs
      CLK,RST                                pin ;
      D15,D14,D13,D12,D11,D10,D9,D8          pin ;
      D7,D6,D5,D4,D3,D2,D1,D0                pin ;
"Outputs
      Q15,Q14,Q13,Q12,Q11,Q10,Q9,Q8          pin istype 'buffer';
      Q7,Q6,Q5,Q4,Q3,Q2,Q1,Q0                pin istype 'buffer';

      B12,B13,B14,B15,B16,B22,B23,B24,B25,B26       node ;
      C2,C5,C7,C9,C11,C13                           node ;

      DIN  = [D15,D14,D13,D12,D11,D10,D9,D8,D7,D6,D5,D4,D3,D2,D1,D0];
      QOUT = [Q15,Q14,Q13,Q12,Q11,Q10,Q9,Q8,Q7,Q6,Q5,Q4,Q3,Q2,Q1,Q0];

C14 = D14 & Q14.Q # D14 & (C13#B16) # Q14.Q & (C13#B16);
C12 = D12 & Q12.Q # D12 & C11 # Q12.Q & C11;
C10 = D10 & Q10.Q # D10 & C9 # Q10.Q & C9;
C8  = D8 & Q8.Q # D8 & C7 # Q8.Q & C7;
C6  = D6 & Q6.Q # D6 & C5 # Q6.Q & C5;
C3  = D3 & Q3.Q # D3 & C2 # Q3.Q & C2;
C4  = D4 & Q4.Q # D4 & C3 # Q4.Q & C3;
C0  = D0 & Q0.Q;
C1  = D1 & Q1.Q # D1 & C0 # Q1.Q & C0;

      H,L,C,X = 1,0,.C.,.X.;

Equations

      QOUT.CLK = CLK;
      QOUT.AR  = RST;

Q15.T = Q15.Q $ (Q15.Q $ D15 $ C14);
Q14.T = Q14.Q $ (Q14.Q $ D14 $ (C13#B16));
Q13.T = Q13.Q $ (Q13.Q $ D13 $ C12);
Q12.T = Q12.Q $ (Q12.Q $ D12 $ C11);
Q11.T = Q11.Q $ (Q11.Q $ D11 $ C10);
Q10.T = Q10.Q $ (Q10.Q $ D10 $ C9);
Q9.T  = Q9.Q $ (Q9.Q  $ D9  $ C8);
Q8.T  = Q8.Q $ (Q8.Q  $ D8  $ C7);
Q7.T  = Q7.Q $ (Q7.Q  $ D7  $ C6);
Q6.T  = Q6.Q $ (Q6.Q  $ D6  $ C5);
```

```
Q5.T  = Q5.Q $ (Q5.Q  $ D5  $ C4);
Q4.T  = Q4.Q $ (Q4.Q  $ D4  $ C3);
Q3.T  = Q3.Q $ (Q3.Q  $ D3  $ C2);
Q2.T  = Q2.Q $ (Q2.Q  $ D2  $ C1);
Q1.T  = Q1.Q $ (Q1.Q  $ D1  $ C0);
Q0.T  = Q0.Q $ (Q0.Q  $ D0 );

B12=(Q5.Q&D5
        "D5&Q5.Q#(D5#Q5.Q)&(D4&Q4.Q)#(D5#Q5.Q)&(D4#Q4.Q)&(D3&Q3.Q)
    # Q5.Q & D4 & Q4.Q
    # D5 & D4 & Q4.Q
    # Q5.Q & D4 & D3 & Q3.Q
    # D5 & D4 & D3 & Q3.Q
    # Q5.Q & Q4.Q & D3 & Q3.Q
    # D5 & Q4.Q & D3 & Q3.Q);

B13 = (Q7.Q & D7              "D7&Q7.Q#(D7#Q7.Q)&(D6&Q6.Q)
    # Q7.Q & D6 & Q6.Q
    # D7 & D6 & Q6.Q);

B14 = (Q9.Q & D9
    # Q9.Q & D8 & Q8.Q
    # D9 & D8 & Q8.Q);          "D11&Q11.Q#(D11#Q11.Q)&(D10&Q10.Q)
B15 = (Q11.Q & D11
    # Q11.Q & D10 & Q10.Q
    # D11 & D10 & Q10.Q);

B16 = (Q13.Q & D13            "D13&Q13.Q#(D13#Q13.Q)&(D12&Q12.Q)
    # Q13.Q & D12 & Q12.Q
    # D13 & D12 & Q12.Q);

B22 = (Q5.Q & D4 & D3         "(D5#Q5.Q)&(D4#Q4.Q)&(D3#Q3.Q)
    # D5 & D4 & D3
    # Q5.Q & Q4.Q & D3
    # D5 & Q4.Q & D3
    # Q5.Q & D4 & Q3.Q
    # D5 & D4 & Q3.Q
    # Q5.Q & Q4.Q & Q3.Q
    # D5 & Q4.Q & Q3.Q);

B23 = (Q7.Q & D6              "(D7#Q7.Q)&(D6#Q6.Q)
    # D7 & D6
    # Q7.Q & Q6.Q
    # D7 & Q6.Q);

B24 = (Q9.Q & D8              "(D9#Q9.Q)&(D8#Q8.Q)
    # D9 & D8
    # Q9.Q & Q8.Q
```

```
    # D9 & Q8.Q);

B25 = (Q11.Q & D10          "(D11#Q11.Q)&(D10#Q10.Q)
    # D11 & D10
    # Q11.Q & Q10.Q
    # D11 & Q10.Q);

B26 = (Q13.Q & D12          "(D13#Q13.Q)&(D12#Q12.Q)
    # D13 & D12
    # Q13.Q & Q12.Q
    # D13 & Q12.Q);

C2 = (Q2.Q & D2
    # Q2.Q & D1 & Q1.Q
    # D2 & D1 & Q1.Q
    # Q2.Q & D1 & D0 & Q0.Q
    # D2 & D1 & D0 & Q0.Q
    # Q2.Q & Q1.Q & D0 & Q0.Q
    # D2 & Q1.Q & D0 & Q0.Q);

C5 = (C2 & B22
    # B12);

C7 = (C2 & B22 & B23
    # B12 & B23
    # B13);

C9 = (C2 & B22 & B23 & B24
    # B12 & B23 & B24
    # B13 & B24
    # B14);

C11 = (C2 & B22 & B23 & B24 & B25
    # B12 & B23 & B24 & B25
    # B13 & B24 & B25
    # B14 & B25
    # B15);

C13 = (C2 & B22 & B23 & B24 & B25 & B26
    # B12 & B23 & B24 & B25 & B26
    # B13 & B24 & B25 & B26
    # B14 & B25 & B26
    # B15 & B26);
    "# B16);


end
```

Memory mapped I/O address decoder

```
Module p9

" Memory mapped I/O address decoder -3 reps
" This is simular to Prep 9

"Inputs
      CLK,RST,AS     pin ;
      A15,A14,A13,A12,A11,A10,A9,A8          pin ;
      A7,A6,A5,A4,A3,A2,A1,A0                pin ;
"Outputs
      A,B,C,D,E,F,G,H,BE                                    pin istype 'buffer,reg_d';
      A0_1,A1_1,A2_1,A3_1,A4_1,A5_1,A6_1,A7_1,AS_1          node istype 'buffer,reg_d';
      A0_2,A1_2,A2_2,A3_2,A4_2,A5_2,A6_2,A7_2,AS_2          node istype 'buffer,reg_d';


         DECODE_1 = [A0_1,A1_1,A2_1,A3_1,A4_1,A5_1,A6_1,A7_1];
      INP = [A15,A14,A13,A12,A11,A10,A9,A8,A7,A6,A5,A4,A3,A2,A1,A0];
                                                                  INP1          =
[A15,A14,A13,A12,A11,A10,A9,A8,A7_1.Q,A6_1.Q,A5_1.Q,A4_1.Q,A3_1.Q,A2_1.Q,
            A1_1.Q,A0_1.Q];
                                                                  INP2          =
[A15,A14,A13,A12,A11,A10,A9,A8,A7_2.Q,A6_2.Q,A5_2.Q,A4_2.Q,A3_2.Q,A2_2.Q,
            A1_2.Q,A0_2.Q];

      L,c,X = 0,.C.,.X.;

Equations

      [A,B,C,D,E,F,G,H,BE].CLK = CLK;
      [A,B,C,D,E,F,G,H,BE].AR  = !RST;
      [A0_1,A1_1,A2_1,A3_1,A4_1,A5_1,A6_1,A7_1,AS_1].CLK = CLK;
      [A0_1,A1_1,A2_1,A3_1,A4_1,A5_1,A6_1,A7_1,AS_1].AR  = !RST;
      [A0_2,A1_2,A2_2,A3_2,A4_2,A5_2,A6_2,A7_2,AS_2].CLK = CLK;
      [A0_2,A1_2,A2_2,A3_2,A4_2,A5_2,A6_2,A7_2,AS_2].AR  = !RST;

      A0_1.D = AS & A15 & A14 & A13 & A12;
      A1_1.D = AS & A15 & A14 & A13 & !A12 & A11;
      A2_1.D = AS & A15 & A14 & A13 & !A12 & !A11 & A10;
      A3_1.D = AS & A15 & A14 & A13 & !A12 & !A11 & !A10 & A9 & A8;
      A4_1.D = AS & A15 & A14 & A13 & !A12 & !A11 & !A10 & A9 & !A8
               & A7 & A6;
      A5_1.D = AS & A15 & A14 & A13 & !A12 & !A11 & !A10 & A9 & !A8
         & A7 & !A6 & A5 & A4;
      A6_1.D = AS & A15 & A14 & A13 & !A12 & !A11 & !A10 & A9 & !A8
         & A7 & !A6 & A5 & !A4 & A3 & A2;
```

```
A7_1.D = AS & A15 & A14 & A13 & !A12 & !A11 & !A10 & A9 & !A8
   & A7 & !A6 & A5 & !A4 & A3 & !A2 & A1 & A0;

AS_1.D = AS & (INP < ^HE2AB) # !AS & AS_1.Q;

A0_2.D = AS_1.Q & A15 & A14 & A13 & A12;
A1_2.D = AS_1.Q & A15 & A14 & A13 & !A12 & A11;
A2_2.D = AS_1.Q & A15 & A14 & A13 & !A12 & !A11 & A10;
A3_2.D = AS_1.Q & A15 & A14 & A13 & !A12 & !A11 & !A10 & A9 & A8;
A4_2.D = AS_1.Q & A15 & A14 & A13 & !A12 & !A11 & !A10 & A9 & !A8
          & A7_1.Q & A6_1.Q;
A5_2.D = AS_1.Q & A15 & A14 & A13 & !A12 & !A11 & !A10 & A9 & !A8
          & A7_1.Q & !A6_1.Q & A5_1.Q & A4_1.Q;
A6_2.D = AS_1.Q & A15 & A14 & A13 & !A12 & !A11 & !A10 & A9 & !A8
          & A7_1.Q & !A6_1.Q & A5_1.Q & !A4_1.Q & A3_1.Q & A2_1.Q;
A7_2.D = AS_1.Q & A15 & A14 & A13 & !A12 & !A11 & !A10 & A9 & !A8
   & A7_1.Q & !A6_1.Q & A5_1.Q & !A4_1.Q & A3_1.Q & !A2_1.Q & A1_1.Q &
A0_1.Q;

AS_2.D = AS_1.Q & (INP1 < ^HE2AB) # !AS_1.Q & AS_2.Q;

A.D = AS_2.Q & A15 & A14 & A13 & A12;
B.D = AS_2.Q & A15 & A14 & A13 & !A12 & A11;
C.D = AS_2.Q & A15 & A14 & A13 & !A12 & !A11 & A10;
D.D = AS_2.Q & A15 & A14 & A13 & !A12 & !A11 & !A10 & A9 & A8;
E.D = AS_2.Q & A15 & A14 & A13 & !A12 & !A11 & !A10 & A9 & !A8
          & A7_2.Q & A6_2.Q;
F.D = AS_2.Q & A15 & A14 & A13 & !A12 & !A11 & !A10 & A9 & !A8
          & A7_2.Q & !A6_2.Q & A5_2.Q & A4_2.Q;
G.D = AS_2.Q & A15 & A14 & A13 & !A12 & !A11 & !A10 & A9 & !A8
          & A7_2.Q & !A6_2.Q & A5_2.Q & !A4_2.Q & A3_2.Q & A2_2.Q;
H.D = AS_2.Q & A15 & A14 & A13 & !A12 & !A11 & !A10 & A9 & !A8
           & A7_2.Q & !A6_2.Q & A5_2.Q & !A4_2.Q & A3_2.Q & !A2_2.Q &
A1_2.Q & A0_2.Q;
BE.D = AS_2.Q & (INP2 < ^HE2AB) # !AS_2.Q & BE.Q;


end
```

8-Bit Fast Parity Generator

```
Module  parity

"bwb

Title   '8 bit fast parity generator';

        D7..D0          Pin;                    "Data to check for parity
        Odd             Pin     istype 'com';   "Output of generator
        x5,x6           node    istype 'com';   "Intermediate parity check results

        x1 = D0 $ D1;           "Exclusive or of first set of data bits
        x2 = D2 $ D3;           "Exclusive or of second set of data bits
        x3 = D4 $ D5;           "Exclusive or of third set of data bits
        x4 = D6 $ D7;           "Exclusive or of fourth set of data bits

equations

        x5  = x1 $ x2;          "Exclusive or of first and second sets of data bits
        x6  = x3 $ x4;          "Exclusive or of third and fourth sets of data bits
        Odd = x5 $ x6;          "Output equals exclusive or of all exclusive ors

end
```

DRAM Refresh Counter

```
Module  refresh
Title   'This is a DRAM Refresh Counter for a  DRAM Controller. This DRAM
        Refresh Counter counts up to 256 states which at 25 MHz clock
        frequency (40 nanosecond cycle period) consists of a time period
        of 10 microseconds. This DRAM refresh counter can be easily modified
        to any desired time period by adjusting the number of states in this
        counter.'


MPCLK, RFSTARTn, MPRSTn, OEn                        pin;
RFUSMST3, RFUSMST2, RFUSMST1, RFUSMST0              pin  istype 'reg';
RFLSMST3, RFLSMST2, RFLSMST1, RFLSMST0              pin  istype 'reg';
RFTIMEn                                             pin  istype 'reg';

rfust = [RFUSMST3, RFUSMST2, RFUSMST1, RFUSMST0];
rfust0   = ^h0;
rfust1   = ^h1;
rfust2   = ^h3;
rfust3   = ^h7;
rfust4   = ^hf;
rfust5   = ^he;
rfust6   = ^hc;
rfust7   = ^h8;
rfust8   = ^h9;
rfust9   = ^hb;
rfust10  = ^ha;
rfust11  = ^h2;
rfust12  = ^h6;
rfust13  = ^h4;
rfust14  = ^h5;
rfust15  = ^hd;

rflst = [RFLSMST3, RFLSMST2, RFLSMST1, RFLSMST0];
rflst0   = ^h0;
rflst1   = ^h1;
rflst2   = ^h3;
rflst3   = ^h7;
rflst4   = ^hf;
rflst5   = ^he;
rflst6   = ^hc;
rflst7   = ^h8;
rflst8   = ^h9;
rflst9   = ^hb;
rflst10  = ^ha;
rflst11  = ^h2;
rflst12  = ^h6;
```

```
rflst13  = ^h4;
rflst14  = ^h5;
rflst15  = ^hd;

        x = .x.; "Don't Care
        z = .z.; "High Impedance
        c = .c.; "Clock
        u = .u.; "Undefined

equations

rfust.C         = !MPCLK;
rfust.OE        = !OEn;
rfust.AR        = !MPRSTn;

rflst.C         = !MPCLK;
rflst.OE= !OEn;
rflst.AP = !MPRSTn;

RFTIMEn.C    = !MPCLK;
RFTIMEn.OE   = !OEn;
RFTIMEn.AP   = 0;

!RFTIMEn     := ((rfust== rfust15)  &  MPRSTn);


STATE_DIAGRAM[RFUSMST3, RFUSMST2, RFUSMST1, RFUSMST0]

state rfust0:    if (!RFSTARTn) then rfust0
                 else if (rflst == rflst15) then rfust1
                 else rfust0;

state rfust1:    if (!RFSTARTn) then rfust0
                 else if (rflst == rflst15) then rfust2
                 else rfust1;

state rfust2:    if (!RFSTARTn) then rfust0
                 else if (rflst == rflst15) then rfust3
                 else rfust2;

state rfust3:    if (!RFSTARTn) then rfust0
                 else if (rflst == rflst15) then rfust4
                 else rfust3;

state rfust4:    if (!RFSTARTn) then rfust0
                 else if (rflst == rflst15) then rfust5
                 else rfust4;
```

```
state rfust5:    if (!RFSTARTn) then rfust0
                 else if (rflst == rflst15) then rfust6
                 else rfust5;

state rfust6:    if (!RFSTARTn) then rfust0
                 else if (rflst == rflst15) then rfust7
                 else rfust6;

state rfust7:    if (!RFSTARTn) then rfust0
                 else if (rflst == rflst15) then rfust8
                 else rfust7;

state rfust8:    if (!RFSTARTn) then rfust0
                 else if (rflst == rflst15) then rfust9
                 else rfust8;

state rfust9:    if (!RFSTARTn) then rfust0
                 else if (rflst == rflst15) then rfust10
                 else rfust9;

state rfust10:   if (!RFSTARTn) then rfust0
                 else if (rflst == rflst15) then rfust11
                 else rfust10;

state rfust11:   if (!RFSTARTn) then rfust0
                 else if (rflst == rflst15) then rfust12
                 else rfust11;

state rfust12:   if (!RFSTARTn) then rfust0
                 else if (rflst == rflst15) then rfust13
                 else rfust12;

state rfust13:   if (!RFSTARTn) then rfust0
                 else if (rflst == rflst15) then rfust14
                 else rfust13;

state rfust14:   if (!RFSTARTn) then rfust0
                 else if (rflst == rflst15) then rfust15
                 else rfust14;

state rfust15:   if (!RFSTARTn) then rfust0
                 else rfust15;


STATE_DIAGRAM[RFLSMST3, RFLSMST2, RFLSMST1, RFLSMST0]

state rflst0:    if (!RFSTARTn) then rflst0
```

```
                else rflst1;

state rflst1:   if (!RFSTARTn) then rflst0
                else rflst2;

state rflst2:   if (!RFSTARTn) then rflst0
                else rflst3;

state rflst3:   if (!RFSTARTn) then rflst0
                else rflst4;

state rflst4:   if (!RFSTARTn) then rflst0
                else rflst5;

state rflst5:   if (!RFSTARTn) then rflst0
                else rflst6;

state rflst6:   if (!RFSTARTn) then rflst0
                else rflst7;

state rflst7:   if (!RFSTARTn) then rflst0
                else rflst8;

state rflst8:   if (!RFSTARTn) then rflst0
                else rflst9;

state rflst9:   if (!RFSTARTn) then rflst0
                else rflst10;

state rflst10:  if (!RFSTARTn) then rflst0
                else rflst11;

state rflst11:  if (!RFSTARTn) then rflst0
                else rflst12;

state rflst12:  if (!RFSTARTn) then rflst0
                else rflst13;

state rflst13:  if (!RFSTARTn) then rflst0
                else rflst14;

state rflst14:  if (!RFSTARTn) then rflst0
                else rflst15;

state rflst15:  goto rflst0;

end
```

Thunderbird Tailight Control Circuit

```
MODULE tbird

TITLE 'thunderbird tailight control Circuit

BRAKE pin;
TL pin;
TR pin;
CLK pin;

L1..L3 pin istype 'reg,buffer';
R1..R3 pin istype 'reg,buffer';
Q2..Q0 pin istype 'reg,buffer' ;

out = [L3, L2, L1, R1, R2, R3];

sreg = [Q2, Q1, Q0];

s0 = [0, 0, 0] ;
s1 = [0, 0, 1] ;
s2 = [0, 1, 0] ;
s3 = [0, 1, 1] ;
s4 = [1, 0, 0] ;
s5 = [1, 0, 1] ;
s6 = [1, 1, 0] ;
s7 = [1, 1, 1] ;

equations

sreg.clk = CLK ;

out.clk = CLK ;

state_diagram sreg ;

state s0:
  if (TL) then s1
  else if (TR) then s4
  else s0;

state s1:
  if (TL) then s2
  else if (TR) then s4
  else s0;

state s2:
  if (TL) then s3
```

```
  else if (TR) then s4
  else s0;

state s3:
  if (TL) then s0
  else if (TR) then s4
  else s0;

state s4:
  if (TR) then s5
  else if (TL) then s1
  else s0;

state s5:
  if (TR) then s6
  else if (TL) then s1
  else s0;

state s6:
  if (TR) then s0
  else if (TL) then s1
  else s0;

state s7:
  goto s0;          "Illegal state recovery

truth_table ([sreg.fb, BRAKE,  TR,  TL] :> [L3, L2, L1, R1, R2, R3])
              [      s0   ,   0  , .x., .x.] :> [ 0,  0,  0,  0,  0,  0];
              [  s0   ,   1  ,   0,   0] :> [ 1,  1,  1,  1,  1,  1];
              [  s0   ,   1  ,   0,   1] :> [ 0,  0,  0,  1,  1,  1];
              [  s0   ,   1  ,   1,   0] :> [ 1,  1,  1,  0,  0,  0];
              [  s1   ,   0  , .x., .x.] :> [ 0,  0,  1,  0,  0,  0];
              [  s1   ,   1  , .x., .x.] :> [ 0,  0,  1,  1,  1,  1];
              [  s2   ,   0  , .x., .x.] :> [ 0,  1,  1,  0,  0,  0];
              [  s2   ,   1  , .x., .x.] :> [ 0,  1,  1,  1,  1,  1];
              [  s3   ,   0  , .x., .x.] :> [ 1,  1,  1,  0,  0,  0];
              [  s3   ,   1  , .x., .x.] :> [ 1,  1,  1,  1,  1,  1];
              [  s4   ,   0  , .x., .x.] :> [ 0,  0,  0,  1,  0,  0];
              [  s4   ,   1  , .x., .x.] :> [ 1,  1,  1,  1,  0,  0];
              [  s5   ,   0  , .x., .x.] :> [ 0,  0,  0,  1,  1,  0];
              [  s5   ,   1  , .x., .x.] :> [ 1,  1,  1,  1,  1,  0];
              [  s6   ,   0  , .x., .x.] :> [ 0,  0,  0,  1,  1,  1];
              [  s6   ,   1  , .x., .x.] :> [ 1,  1,  1,  1,  1,  1];

END
```

Denver International Air Traffic Controller

```
MODULE traffic1

TITLE 'Denver International Air Traffic Controller'

" The output is not that of a conventional traffic light controller

clk pin ;
sena pin ;
senb pin ;
rst pin ;
ga, ya, ra pin istype 'reg';
gb, yb, rb pin istype 'reg';

s3..s0 node istype 'reg' ;

h, l, ck, x = 1, 0, .c., .x. ;
count = [s3..s0] ;

on = [1, 0] ;
off = [0, 1] ;

equations

[gb, yb, rb].ar = rst ;
[ga, ya, ra].ar = rst ;
[gb, yb, rb].clk = clk ;
[ga, ya, ra].clk = clk ;

[s3..s0].ar = rst ;
[s3..s0].clk = clk ;

state_diagram count

        state 0:  if (sena & !senb) then 0
                        else if (!sena & senb) then 4
                        else if (sena == senb) then 1 ;

        state 1:  goto 2 ;
        state 2:  goto 3 ;
        state 3:  goto 4 ;
        state 4:  ga = off ;
                        ya = on ;
                        goto 5 ;

        state 5:  ya = off ;
                        ra = on ;
```

```
                    rb = off ;
                    gb = on ;
                    goto 8 ;

state 6:  goto 0 ;
state 7:  goto 0 ;

state 8:  if (!sena & senb) then 8
                    else if (sena & !senb) then 12
                    else if (sena == senb) then 9 ;

state 9:  goto 10 ;
state 10:         goto 11 ;
state 11:         goto 12 ;

state 12:         gb = off ;
                  yb = on ;
                  goto 13 ;

state 13:         yb = off ;
                  rb = on ;
                  ra = off ;
                  ga = on ;
                  goto 0 ;

state 14:         goto 0 ;

state 15:         ra = off ;
                  ya = off ;
                  ga = on ;
                  rb = on ;
                  yb = off ;
                  gb = off ;
                  goto 0 ;


END
```

Appendix D *Error Messages*

This appendix is a list of the errors and warnings provided by XPLA Designer during the parsing, optimizing, and fitting of designs. The first section provides errors related to the parsing of the source file. The error message is an attempt to provide you with constructive guidance in correcting a problem. If you need help in solving a problem, be sure to contact Philips Applications as detailed in Chapter 1.

The second section provides fitter related errors. Generally, errors fall into two categories: design errors and file I/O errors. File I/O errors occur when the fitter either can't open a file to read or write. A fit error can be due to incorrect use of the device, or not enough resources in the device for the design (too many inputs, outputs, registers, product terms,etc.)

In this appendix, errors are given followed by a recommended action. Errors are provided in the following format:

<error type> <number>: <text>

The following error types are used

WARNING	####	A warning may or may not be a problem.
LOGICAL ERROR	####	A design error.
SYNTAX ERROR	####	A language syntax error
COMMAND ERROR	####	An error in a command line option or augument.
DEVICE ERROR	####	An error related to the device file.
FATAL ERROR	####	An error that causes program termination.
INTERNAL ERROR	####	A program error.

The compiler accepts PHDL, open-ABEL (PLA), and BLIF input formats. There are many tools which write netlists in BLIF and PLA format. When the recommended action is to re-generate the BLIF or PLA file, this may involve editing a source file used in the generation of the file.

Warnings

1010 Normalizing node to active-high.

1020 Restoring node to active-low.

2355 Node with both KEEP and COLLAPSE. KEEP is assumed.

If you want the node collapsed, delete the keep directive.

2357 Creating node ...

2360 Node collapsed.

2365 Node swept.

2381 The device supports maximum pterm for each node between 5 and 37 . Specified as <range> in command line. Changed to default.

This parameter affects the speed and density of the fitting. If designs objectives are not met, this number should be varied, and the design re-compiled and re-fit.

2382 The device supports maximum fanin for each node between 1 and 36. Specified as <number> in command line. Changed to default value.

The default value is 36.

2385 Fanin of node must between 1 and device limit. Changed from <number> to default.

The fanin is changed to 36.

2401 Network node does not fanout.

If this node is intended to be used, modify the PHDL code.

2403 Network node is not driven (PI assumed).

If this node is intended to be used in the design, edit the PHDL code.

2405 The following signal(s) form a combinational loop:

Break the combinatorial loop in the PHDL code.

2420 Preset and reset can not both be assigned to a signal.

Please delete either the preset or reset port attached to the signal.

2430 Equation with INVERT type. Normalized to equivalent active-high logic. Power-up value has been changed.

Verify that polarity of the signal is correct.

2490 Test vectors are not supported. This section is ignored!

The design can be simulated using XPLA Designer.

2500 Warning near line <number>. Directive \"@\" not supported in this release. Ignored!

This directive is not supported in PHDL.

2510 Trace statement not supported in this release. Sections ignored!

The design can be simulated using XPLA Designer

2600 Control signals has <number> pterms. Converted to one.

Verify that the functionality of the control term is as intended.

5180 line : NOT (!) operator is ignored in port connection.

Verify the functionality of the port is the correct polarity. If not, re-name the signal and re-write the right hand side of the equation.

5190 line : In hierarchial design, a module cannot have a default value.

Remove default values in the interface and/or functional block records in the PHDL source.

5195 Output port does not have a connection.

Verify that the output is unused. If the output is used, add logic to connect the port.

5202 : A sub-module cannot have a default value.

Corect the interface and/or functional block records in the top-level PHD source, and the module statement in the sub-module.

5205 Line <number> : Default value is ignored in this module !!!

Corect the interface and/or functional block records in the top-level PHD source, and the module statement in the sub-module.

ERRORS

108 : The command line option must be a number.

A string is found. Replace the string with a number.

118 : No argument after parameter. A number is required.

Add a number after the parameter in the command line.

120 : Device not supported.

Either select a device from the cyclic field in the XPLA Designer or verify that the correct part number is used in the command line. The part number syntax is given in the data sheet.

150 Cannot open file.

Check the location and permissions on the file and the directory being read/written.

200

Contact Philips Applications.

1005 Unknown command line option found.

Use a valid command line option.

2300 Error in BLIF file <file> at line <number>. Unknown character . .

Regenerate the BLIF file.

2302 Unknown token in BLIF file at line <number>.

Regenerate the BLIF file.

2305 Syntax error in file <file> near line <number>.

Regenerate the BLIF file.

2315 File <file> line <number> : error in pla format.

Regenerate pla file and verify that there are no errors. See p 3-1 to 3-17 of Open-Abel Technical Specification for correct syntax of the pla format supported by XPLA Designer .

2317 File <file> line <number> : error ".names" list.

There is an error in the BLIF source. Regenerate the BLIF file.

2318 File <file> line<number> : error fanin number.

In the PLA or BLIF source, the number of fanins listed is different than the number of fanins. Regenerate the BLIF file.

2320 File <file> line <number> : Output can only has one digit, <number> is found.

The output values in PLA or BLIF covers can have a value of 1 or 0. Regenerate the BLIF or PLA file.

2321 File <file> line <number> : Error output format , must be "1" or "0".

In PLA or BLIF format, outputs in the covers must be 1 or 0. Regenerate the PLA or BLIF file.

2322 File <file> line <number> : Output has been previously defined.

The output in the PLA or BLIF file is listed more than once. Regenerate the PLA or BLIF file.

2324 Input has been previously defined.

The input in the PLA or BLIF file is listed more than once. Regenerate the PLA or BLIF file.

2326 Error near line <number>. Signal multiply defined.

The signal in the PLA file has already been defined. Regenerate the PLA file.

2328 Signal is not declared near line <number>.

The signal in the PLA file is not declared. Regenerate the PLA file.

2330 File <file> line <number> : Module already defined.

The module in PHDL file has been defined. Edit the PHDL file.

2335 Syntax error in file near line <number>.

Correct the syntax in the PHDL file. Refer to chapters 5 and 8 of this manual for correct syntax of PHDL.

2340 Syntax error near line <number>. Extension <string> is not allowed in output.

Correct the PHDL file by deleting the extension on the output.

2345 File <file> line <number> : unknown token .

Correct the PHDL file. Supported tokens, keywords, and identifiers are provided on p 5-1 and 5-2 of this manual.

2350 File <file>: Only "0", "1", or "-" is allowed in pterm. <string> is found.

Regenerate the PLA or BLIF to change table to a valid table.

2370 Node not in PXAPIN list.

Contact Philips Applications.

2375 Pxapin not in node list of pxainfo.

Contact Philips Applications

2377 Pxapin not in pin_table of pxainfo.

Contact Philips Applications.

2435 Register equation can not have COM feedback.

Correct the dot extension in the PHDL source. See table 5-4 for a list of valid dot extensions.

2437 Node equations can not have PIN feedback.

Correct the dot extension in the PHDL soure. See table 5-4 for a list of valid dot extensions.

2440 Register equation can not have D feedback.

Correct the dot extension in the PHDL source. See table 5-4 for a list of valid dot extensions.

2445 Equation shouldn't appear at normalization. Internal Error.

Please contact Philips Applications.

2460 Number of pin-node list is not the same as number list.

Correct the declaration section of the PHDL source.

2465 line <number> Syntax a1..b1 not allowed in a declaration statement.

The values for a1 and b1 are not compatible for range operator. Correct the PHDL source in the declaration section.

2470 Number <number> is not allowed near line <number>. Only 0 or 1 is allowed.

Revise the source so that either 0 or 1 is used.

2475 Size of set not compatible near line <number> LHS= <number>, RHS=<number>.

Correct the PHDL source so that the expression for the right hand side is the same size as that on the left hand side.

2480 Error near line <number>. <string> is not supported for sets.

Revise the PHDL source. For correct use of sets in PHDL, refer to chapter 5 of this manual.

2485 Syntax error near line <number>. Set is not allowed in WHEN condition.

Revise the PHDL source so that a set is not used in a WHEN conditional statement. One option may be to use discrete signals in the conditional statement.

2495 Constant declaration must be one-to-one correspondence.

Revise the PHDL source so that the constant is defined only once.

2515 Error near line <number>. State exp must all 0/1s in this release.

Revise the PHDL state machine description. See p 43 for an example.

2520 Error near line <number>. Size of state register () is different with state esp ().

Revise the PHDL state machine description. See p. 43 for an example.

2530 Different set size, (LHS = <number>, RHS = <number>) in operation <string>.

Correct the set size. See Chapter 5 for the correct use of sets in PHDL.

3000 PLA type not supported.

XPLA Designer supports only PLA formats compatible with that defined in the Open-Abel Technical Specification.

3002 Number of signal in '.ilb' list () doesn't match '.i' ().

In the PLA file, the number of names in the .ilb record must equal the number in the .i record. Re-generate the PLA file.

3004 Number of signal in '.ob' list () doesn't match '.o' ().

Correct the PLA file such that the number of names in the .ob record is equal to the number in the .o record.

5000 Dulpicate state near line <number>.

Correct the PHDL state machine description. See p43 for an example.

5005 Missing parameters for macro near line <line>.

Correct the macro in the PHDL source. See Chapters 5 and 8 for the correct use of macros in PHDL.

5010 Parameter not declared in macro near line <line>.

Correct the macro in the PHDL source. See Chapters 5 and 8 for the correct use of macros in PHDL.

5015 Recursive usage of macro (near line).

Correct the macro in the PHDL source. See Chapters 5 and 8 for the correct use of macros in PHDL.

5020 The number of parameters is incorrect near line <number> .

Correct the number of parameters in the PHDL source.

5025 Syntax error near line <number> . Extension is not supported.

Correct the PHDL source. Valid dot extensions are listed in Chapter 5.

5030 Entries in truth table do not match with header near line <number>.

Correct the truth table in the PHDL source. See p 44-45 for valid truth table syntax. The record should be a format of:

truth_table ([inputs] -> [output]) ;
[input state n] -> [output state n] ;

5035 Number of variables in truth table do not match with header near line <number>/

Correct the truth table in the PHDL source. See p 44-45 for valid truth table syntax.

5040 Data of truth table are not consistent.

Correct the truth table in the PHDL source. See p 44-45 for valid truth table syntax.

5045 line <number>: Syntax error.

Correct PHDL syntax error.

5050 Espresso : Some minterm(s) belong to both the ON-SET and DC-SET.

Correct error in PLA file.

5052 Espresso : Some minterm(s) belong to both the ON-SET and OFF-SET.

Correct error in PLA file.

5054 Espresso : Some minterm(s) belong to both the OFF-SET and DC-SET.

Correct error in PLA file.

5056 Espresso : There are minterms left unspecified.

Correct error in PLA file.

5060 Line <number> : ';' expected.

Correct error in PHDL file.

5065 Line <number> : Single quotation mark expected.

Correct error in PHDL file.

5070 Line <number> : Module name expected.

Correct error in PHDL file.

5075 Line <number> : '{' or '(' expected.

Correct error in PHDL file.

5080 Line <number> : Pin number or istype or ';' expected.

Correct error in PHDL file.

5085 Line <number> : Device or istype name expected.

Correct error in PHDL file.

5090 Line <number> : ']' expected.

Correct error in PHDL file.

5095 Line <number>: Unknown dot extension ''.

Correct error in PHDL file. See Chapter 5 for valid dot extensions in PHDL.

5100 Line <number> : '(' expected.

Correct error in PHDL file.

5105 Line <number> : ')' expected.

Correct error in PHDL file.

5110 Line <number> : '[' expected.

Correct error in PHDL file.

5115 Line <number> : ELSE expected.

See p 40-41 for correct syntax of conditional statements in PHDL. Correct the PHDL file.

5120 Line <number> : THEN expected.

Correct error in PHDL file.
See p 40-41 for correct syntax of conditional statements in PHDL. Correct the PHDL file.

5125 Line <number>: IF, CASE, or GOTO statement expected.

See p 40-41 for correct syntax of conditional statements in PHDL. Correct the PHDL file.

5130 Line <number> : Possible missing ':' or ';' in CASE statement.

See p 40-41 for correct syntax of conditional statements in PHDL. Correct the PHDL file.

5135 Line <number> : ';' or WITH statement expected.

See p 40-41 for correct syntax of conditional statements in PHDL. Correct the PHDL file.

5140 Line <number> : ENDCASE expected.

See p 40-41 for correct syntax of conditional statements in PHDL. Correct the PHDL file.

5145 Line <number> : Dot extension is not allowed in output.

Correct error in PHDL file.

5150 Line <number> : Dot extension is not allowed in input.

Correct error in PHDL file.

5155 Line <number> : '=' or ':=' expected.

Add the assignment operator to the PHDL file.

5160 Line <number> : Identifier must begin with an alphabetic character.

Change the identifier in PHDL file.

5165 Line <number> : Signal is not declared in interface statement.

Change the interface statement in PHDL file.

5170 Line <number> : Formal and actual signal assignment do not match.

Correct error in PHDL file.

5175 Line <number> : Constant has been declared already.

Ensure that the constant is declared only once in PHDL file.

5185 Line <number> : Submodule has not been declared in INTERFACE statement.

Add the submodule to the interface statement in PHDL file

5200 Line <number> : Instance has not been declared.

Add the interface and/or functional_block statement in top-level PHDL file.

5210 Line <number> : Submodule has been declared already.

Correct the interface and/or functional_block statement in the top-level PHDL file.

5215 Port has not been connected.

Correct the PHDL file.

5220 Line <number> : Instance has been declared already.

Correct the interface and/or functional_block statement in the top-level PHDL file.

5225 Line <number>: Dot extension is not allowed in port name.

Correct the name of the input, output, or inout in the PHDL file.

5230 Line <number> : Primary equation for exists.

Correct the PHDL file.

5235 Line <number> : Input port has already been connected.

Correct the connectivity of the PHDL file.

5240 Input file not found.

Either use the XPLA Designer GUI to navigate to the correct directory or invoke the command line interface form the project directory containing the design file.

5245 Line <number>: Operand needed for operator .

Table 5.2 provides the operators supported in PHDL. Correct the PHDL file.

5250 Line <number> : Id needed for property statement.

Add the identifier in the PHDL file. As an example of the syntax of a property statement in the header section is
xpla property ‘dut on’ ;

5255 Line <number>: ';' needed for macro statement.

Add the semicolon to macro statement in PHDL file. See p 5-X for the correct syntax for using macros.

5260 Line <number

>: syntax error within macro definition.

Correct the macro in the PHDL file. See p 5-x for the correct syntax for using macros.

5265 Line <number> : The name of module has to be the same with its filename without extension.

Correct the module statement in the PHDL file.

5270 Line <number> : Bad istype ' '.

Supported istype attributes in PHDL are listed in Table 5-3 and 5-5. The use of istyoe definition of pin and nodes is discussed on p 5-x.

Fitter Error Messages

Below is a list of the error messages and warnings resulting from XPLA Designer's fit operation. Command and Internal errors indicate that something has gone wrong with the fit program. If you encounter one of these errors, call the technical support line for help. Logical and Fatal errors as well as warnings indicate that there is a problem with your design or the way you are trying to fit the design. These are generally descriptive enough that you can correct the problem and re-compile and fit the design after it has been corrected.

Warnings

3211: Synchronous signal converted to asynchronous.

Indicates that you were trying to drive a synchronous clock with a signal generated in the logic. These signals can only drive asynchronous clocks.

3212: Power-up reset signal is changed to non-inverted output.

When inverted outputs are specified, they will revert to the standard (low) state after power-up reset.

3213: Power-up preset signal is changed to non-inverted output.

When inverted outputs are specified, they will revert to the standard (high) state after power-up preset.

3224: Reset/Preset cannot be connected to V_{DD}.

You should not wire the reset or preset function to V_{DD}. The devices are automatically reset/preset at power up.

3240: Unable to route fixed pinout. Now using automatic pin assignment.

Indicates that the design could not be routed with the specified pinout and that the software automatically unlocked the pins and routed the design.

3242: Unable to partition fixed pinout. Now using automatic pin assignment.

Indicates that the logic could not be partitioned into the logic blocks with the specified pinout and that the software automatically unlocked the pins and routed the design.

3244: <pin name> is a floating node. It will be assigned to a pin.

Indicates that an unused node is an input and that it will be assigned to pin <pin name>.

3248: Pin feedback for signal <signal name> is invalid due to output disable.

Indicates that you are using feedback from pin <signal name> that has an output enable. When the output is off, the logic that uses the feedback will be undefined.

3249: Output enable for NODE <node name> is ignored.

Indicates that buried node <node name> was assigned an output enable which will do nothing because it is not an output.

Fatal Errors

3206: Out of memory.

Your system does not have enough memory to run the process requested. Close other applications and try again.

3207: Identifier table full.

There are too many identifier names. Reduce the number of variable names in your design.

3208: BLIF file is not supported.

The BLIF file you have tried to read in is not supported by the Device Kit.

3209: Unable to open/close <file name>.

The file you are trying to access either has a different name or does not exist.

3214: Invalid device name or not a valid device.

You must use only device names listed in the Device Kit.

3215: <extension> is not a valid extension.

Indicates that the dot extension <extension> is not supported.

3216: Design exceeds maximum number of synchronous clocks.

You are trying to use more synchronous clocks than the targeted device will support.

3217: Clock input cannot be tied to V_{DD}/GND.

You cannot connect clock inputs to power or ground.

3218: Too many product terms for <node name>, maximum is 37.

Node <node name> has more than 37 product terms which is not supported by Philips devices.

3219: Design exceeds maximum number of asynchronous clocks.

You are trying to generate more clocks than the device will support from the logic.

3220: Output enable can be driven by only one product term.

The output enable signal for Philips devices can only be driven by a single product term.

3221: Design exceeds maximum number of output enables.

You are trying to use more output enable terms than the device will support.

3222: Reset/preset can be driven by only one product term.

The reset and preset signals for Philips devices can only be driven by a single product term.

3223: Design exceeds maximum number of resets/presets.

You are trying to use more reset/preset signals than the device will support.

3225: Selected device is not supported by the fitter.

You have specified an incorrect Philips CPLD name or a non-Philips part that is not supported by the Device Kit.

3226: Design exceeds maximum number of pins.

Your design uses a total number of I/O pins and dedicated inputs than the targeted device can support.

3227: Design exceeds maximum number of nodes.

Your design uses a total number of nodes that exceeds the maximum number the targeted device will support.

3228: Design exceeds maximum number of inputs.

Your design uses a total number of inputs greater than what the device will allow.

3236: Software unable to fit design.

The fitter was not able to fit your design into the device using the current fitting property settings.

3237: Unable to KEEP pin assignment.

The pin assignment you used for the last fitting could not be reused.

3239: Unable to route input signal <signal name>.

The fitter was not able to route signal <signal name> where it was needed in the logic.

3243: Preset/reset cannot both be assigned to register <register number>.

Registers in Philips devices can be reset or preset, but not both at the same time.

3245: Single node exceeds maximum allowable fan in.

There is a node in the design that has more signals routed to it than the device can support.

3246: Signal <signal name> cannot be both combinatorial and registered.

Signal declarations must be either combinatorial or registered, not both.

3250: Floating node found-- <node name>.

The fitter found a node (<node name>) that is not defined.

3251: Combinatorial feedback before a register is not a valid feature.

You cannot use the input to a register as feedback for your design.

3252: Inverter after T/JK/SR flip-flop is not synthesizable--<signal name>.

T/JK/SR flip-flops do not support inversion.

3253: Cannot enable GTS. Pin already assigned to <signal name>.

You cannot use the Global Tri-State (GTS) feature because the pin is already assigned to another signal.

Logical Errors

3210: Design exceeds maximum number of flip-flops.

Your design uses a total number of registers that exceeds the maximum number the targeted device will support.

3229: Invalid pin specification or attribute for pin <pin number>.

You have tried to assign an attribute or a signal to a pin that cannot support that type; i.e. an output assigned to a dedicated input.

3230: Pin <pin name> is not a valid clock pin.

You have tried to assign a clock to pin <pin name> and that pin will not support clocking.

3231: Cannot use asynchronous clock. Pin is already assigned.

All of the pins that can support asynchronous clocks have already been assigned to other signals, so you cannot use an asynchronous clock.

3232: Design exceeds maximum allowable fan-in for a logic block.

There is a logic block that has more signals routed to it than the device can support.

3233: Design exceeds maximum number of product terms for a logic block.

You have specified too many logic terms for a single logic block.

3234: Design exceeds maximum number of presets/resets for a logic block.

Only two preset/reset signals are allowed per logic block.

3235: Design exceeds maximum number of output enable terms for a logic block.

You have assigned more output enable terms in one logic block than the part will support.

Command and Internal Errors

3200: Incorrect program name.

3201: Unrecognized argument option for -preassign.

3202: Unable to open input file.

3203: Unrecognized 'switch' option.

3204: '-' is required for command switch.

3205: Parser reading error.

3238: Unable to do pin assignment.

3241: Unknown clock pin found during fuse mapping.

3247: Unknown zia line found.

Appendix E *Software License Agreement*

1. LICENSE. Philips Semiconductors ("Philips") hereby grants you, as a Customer and Licensee, a single-user, non-exclusive license to use the enclosed Philips software program ("Program") on a single CPU at any given point in time. Philips authorizes you to make archival copies of the software for the sole purpose of backing up your software and protecting yourinvestment from loss.

2. TERM AND TERMINATION. This agreement is effective from the date the CDROM or diskettes are received until this agreement is terminated. The unauthorized reproduction or use of the Program will immediately terminate this Agreement without notice. Upon termination you are to destroy both the Program and the documentation.

3. COPYRIGHT AND PROPRIETARY RIGHTS. The Program is protected by both United States Copyright Law and International Treaty provisions. This means that you must treat the Program just like a book, with the exception of making archival copies for the sole purpose of protecting your investment from loss. The Program may be used by any number of people, and may be moved from one computer to another, so long as there is **No Possibility** of its being used by two people at the same time.

4. DISCLAIMER. THIS PROGRAM AND DOCUMENTATION ARE LICENSED "AS-IS," WITHOUT WARRANTY AS TO PERFORMANCE. PHILIPS EXPRESSLY DISCLAIMS ALL WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTY OF MERCHANTABILITY OR FITNESS OF THIS PROGRAM FOR A PARTICULAR PURPOSE.

5. LIMITED WARRANTY. The CDROM or diskettes on which this Program is recorded is guaranteed for 90 days from date of purchase. If a defect occurs within 90 days, contact the representative at the place of purchase to arrange for a replacement.

6. LIMITATION OF REMEDIES AND LIABILITY. IN NO EVENT SHALL PHILIPS BE LIABLE FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES RESULTING FROM PROGRAM USE, EVEN IF PHILIPS HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. PHILIPS' EXCLUSIVE LIABILITY AND YOUR EXCLUSIVE REMEDY WILL BE IN THE REPLACEMENT OF ANY DEFECTIVE CDROM OR DISKETTE AS PROVIDED ABOVE. IN NO EVENT SHALL PHILIPS' LIABILITY HEREUNDER EXCEED THE PURCHASE PRICE OF THE SOFTWARE.

7. ENTIRE AGREEMENT. This agreement constitutes the sole and complete Agreement between Philips and the Customer for use of the Program. Changes to this Agreement may be made only by written mutual consent.

8. GOVERNING LAW. This Agreement shall be governed by the laws of the State of California.

Appendix Z *About this manual*

This manual was written in its entirety by the Philips Programmable Products Applications group. Originally we planned to use a professional technical writer to produce this documentation, but a number of us had suffered grievously through trying to use documentation that was written by people who had never used the product (and in fact had little clue about what the product does). Thus we chose to suffer grievously through the process of creating this manual instead of taking the easy way out in the hope of producing something that would be useful to **real** engineers.

This manual was written using MS Word 6.0a, and most sections were created running Windows™ 3.11. The entire thing was compiled and final edited using MS Word 6.0a running on Windows95™. The editor would like to thank Bill and his minions at Microsoft ™ for releasing an OS that can actually open a document file that is larger than 10Mb without crashing. Adobe Acrobat™ 2.1 was used to generate the .pdf file to allow paperless distribution of this manual.

We thought it would be nice to let you know a little about the people that created this tome. Below are what they submitted, written in their own words. Maybe this wasn't such a good idea after all.

The Editor

Mark Aaldering is the Applications and Architecture Development Manager for Philips Programmable Products group. He has held various positions in the Programmable Logic

Field for over ten years. In the creation of this manual, Mark was responsible for the final editing of all of the real work done by the Applications Staff. Asked to comment on this, he replied something about 100 monkeys given typewriters creating a work of literature. In his spare time, Mark continues to work out on that expensive home gym he bought as a New Years resolution (Really! I still use it!) so that his legs will transform into massive, muscle-rippled, tree trunk-like appendages that will allow him to continue kicking his friend Darryl's butt skiing the NASTAR course. The recent breakthrough he made in his garage in working cold fusion is rumored to be ready for publication in a prominent scientific journal, or the National Enquirer, whichever will fork over the bucks first.

The Writers

Chris Schell

Reno Sanchez

B. Wade Baker

Lester Sanders

B. Wade Baker - We asked Wade to tell us a little about himself. Here, in his own words, is what he said. “My name is B. Wade but I go by Bubba. I think the B stands for Bubba but I aint never heard. I was born in Nashville Tennessee and lived at home until the revenuers come an took my maw and paw away.” We asked him about college and he said, “Yeah, I reckon I heard that there is such things but I aint never bothered with em myself.” On the subject of work experience ‘Bubba’ tells us, “I reckon the closest related experience would be the two weeks I worked as a salesman for Radio Shack. Next would probably be the Dairy Queen and the pig slaughter house. I don’t think anything else would be related to what I’m a doin now.” When asked about family, Bubba said, “Well, me and Loretta been married for near bout 15 years. Her father has let on, now that we been married for sooo long, how that the shotgun really wasn’t loaded. We got us a real nice doublewide on the east side with pink flamingos and everthing!” “What do you do when your not working Bubba” we said. Bubba replied, “I reckon my favorite hobby is tryin to convert them there new fangled bucky balls into funky balls so I can use em to amplify my square dance fiddle. After that I reckon it would be watchin wraslin on TV. I heard people a sayin that its fake, but I KNOW it AINT!”

Reno Sanchez - Reno’s educational background includes a BSEE, an MSEE, and an Executive MBA. Before coming to Philips, Reno worked for a large telecommunications

company. It was rumored that Reno left his former job because "climbing telephone poles was becoming too dangerous". Reno's work experience related to manual writing includes reading stories to his children and a third grade Spelling Bee contest (of course, he bombed out after the second round because he spelled "MOM" backwards!). In his spare time, Reno continues to try his luck on the pro rodeo circuit but so far it hasn't managed to knock any sense into him. Reno remains hopeful that he can use his steer wrestling techniques on difficult customers. Reno also enjoys deep sea diving but has had little opportunity to enjoy this activity in New Mexico. Reno always notifies the local "Search and Rescue Authorities" before pursuing this activity in "The Land of Enchantment".

Lester Sanders, an industry veteran, wrote the bulk of this manual, and graciously shares credits with the parasites also in the picture. After all, Luc Longley also got a ring. But think about it. The biker looking dude is Chris Schell. While his new beard isn't as permanent as Mr. Gorbachev's spot, he still hasn't shaven. Mr Baker is still a fan of Philips early DOS-based software called SNAP - some people are easily amused. Any mistakes or questions on sections which read as they were written by an amateur should be directed to Mr. Sanchez. Any reader managing to get help from Mr. Sanchez should contact Mr. Aaldering. Any reader managing to get water out of a rock should contact the **true** author of this manual.

Chris Schell - Barely 17, Chris is the youngest and most congenial member of our group. He tells us: "I attribute my fine customer service to the unique brand of people skills I learned when I was a professional wrestler. In fact, I used those skills to convince Bubba that pro wrestling is indeed real." New to the semiconductor industry, Chris enjoys the travel to exotic places like Omaha and Buffalo. "I don't think there are any motels with 'magic fingers' in Albuquerque", he explains; "I had know idea how big the world really is". He hopes to someday take the family with him on a business trip so they too can enjoy the posh life of a Philips employee. Chris moved from our petroleum division to our group about a year ago. He claims he has more leisure time now that he doesn't have to work anymore double shifts as a service attendant. When not at work, Chris works at finishing school and races cockroaches. "The ones from Florida are the best, but it takes a while to get them altitude trained", he claims.[6]

[6] ***CAUTION:*** *Track 1 on the XPLA Designer CDROM is a Data Track - Do Not Attempt to play this track on a CD Audio player - Damage to your audio system may result.* 'Secret Audio Tracks' courtesy of Philips' FAE Mike Hummel.